UNIVERSITY OF OSLO
Department of Informatics

Design of A Global Multicast Demonstrator for Live Video Streaming

Master Thesis (60 credits)

Liping Huang

May 2, 2013

Design of A Global Multicast Demonstrator for Live Video Streaming

Liping Huang

May 2, 2013

Abstract

Video streaming service has gained more and more popularity in today's Internet. With increasing number of users and video quality, the Internet traffic for video streaming has increased dramatically. At present, most video streaming services utilize server-client model based on unicast. In a unicast system, the servers suffer from overload and lack of scalability with increased video traffic. Multicast has been considered as an efficient communication mechanism and the basis of a decentralized model. It has been proposed and implemented in video streaming service to overcome the drawbacks in unicast systems.

In this work, a global multicast demonstrator for live video streaming is built to investigate how multicast can improve network efficiency. A series of experiments using the demonstrator are performed in real world Internet. The Internet traffic is measured and analyzed. The results show that multicast can reduce the load on servers and the traffic in network.

Acknowledgements

I would like to thank Trarik Cicic, Carsten Griwodz and Stein Gjessing who have inspired, guided and given support during the course of work. I would like to thank especially Tarik Cicic and his company - Media Network Services AS, providing access to a server on the multicast-enabled network and helping with the experiments.

Finally, I would like to thank my family, especially my husband Gang Lin, my two little princes Frank and Sam,

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In recent years, video streaming is the fastest growing application of today's Internet. The report from Cisco[8] figures out that Internet video traffic counts for 51% of global consumer Internet traffic in 2011. There are many prominent examples of video streaming services available in current Internet, such as YouTube[1] and Netflix[2]. In such examples, there are thousands, even millions of users to access video in the same time. The video quality in these video streaming services is also required to near TV resolution standards such as 640×480 pixels for enhanced-definition TV (EDTV) and 1280×720 pixels for high-definition TV (HDTV). As a result, the solutions for video steaming services over Internet must have the ability to support heavy downstream traffic due to many concurrent users and high quality video.

Currently, one of the major solutions for video streaming services is Content Delivery Network (CDN) such as Akamai[3]. It provides a large distributed system of servers across Internet to server users with high availability and high performance. It utilizes the traditional server-client model based on unicast. In such unicast systems, the performance will deteriorate and the servers can be easily overloaded due to increasing number of concurrent users and higher video quality.

To solve this problem and increase the scalability, other communication models are proposed and implemented for video streaming.One of those models is multicast. Multicast refers to the one-to-many or many-to-many communication. Unlike the traditional server-client model based on unicast, the servers in a multicast system do not send each packet repetitively to all clients. The multicast system can reduce the need for the centralized servers. In the beginning, multicast is purposed to be implemented at the network layer[9] and named as Internet Protocol (IP) multicast. In an IP multicast system, the server sends each packet only once even though there are many clients. Each packet is replicated and forwarded on each router, then arrives at a client. IP multicast obviously can handle increasing number of clients without increas-

[1] http://youtube.com/
[2] http://netflix.com/
[3] http://akamai.com/

ing load on the servers. However, it increases the complexities at the network layer and has no support for high-level functionalities such as security, access control and network management, etc[10]. In real world, IP multicast can only work in closed networks such as enterprise networks.

Multicast has also been proposed to be implemented at the application layer[4][7][16] and named as Application Layer Multicast (ALM) in its early stage. In ALM, each host takes over the tasks of routers in IP multicast, such as replicating and forwarding packets. Conviva[4], a pioneer to implement ALM as the solution to video streaming, has developed and deployed the live streaming system based on overlay multicast streaming protocols[6][27][29]. ALM provides low latency and in-order delivery of packets. However, the traffic depends on the multicast tree consisting of clients. It is very sensitive to node failure in its tree. ALM fits into the definition of Peer-to-Peer (P2P) system. The hosts are connected using the virtual links over the overlay network. Some P2P applications such as BitTorrent for file-sharing have gained tremendous popularity and business success. P2P has been proposed and implemented as the solutions for video streaming[17][34][15]. The early ALM based on the multicast tree is also known as tree-based P2P multicast. Most of current P2P video streaming systems have utilized a mesh topology and are known as mesh-based P2P multicast.

Compared to IP multicast, P2P multicast can be implemented in existing network architecture and provide support for high-level functionalities. But it has introduced control overheads and has higher latency than IP multicast. The combination of IP multicast and P2P multicast can take advantage of the primitive solutions while avoid their drawbacks in the same time, such as Universal Multicast framework[33].

1.1 Goal and Contribution

The goal of this thesis is to implement a prototype demonstrator that provides live streaming services using both IP multicast and P2P multicast. This demonstrator will be studied and evaluated in real world Internet. The following contributions are stated in this thesis:

- A literature survey about multicast communication mechanism is presented, with focusing on their key concepts and their opportunities in video streaming.

- Several media streaming systems available in current market are investigated. Some media streaming protocols and P2P streaming technologies are described in detail.

[4]http://conviva.com/

- A prototype demonstrator is implemented on Adobe's Flash platform. This work includes the setup of server and the programming of client applications.
- Experiments using the prototype demonstrator are performed in different network environments such as home, academic and commercial network environments in real world. The Internet traffic related to video streaming are measured and analyzed.

1.2 Outline

The rest of this thesis is organized as follows:

- Chapter 2: Relevant literature review about multicast communication mechanism is presented. Several media streaming systems and the related streaming technologies are described, with focusing on Adobe's media streaming systems.
- Chapter 3: The technologies and tools on Adobe's Flash platform are presented. The detailed implementations on server's and client's side are described.
- Chapter 4: A series of experiments using the prototype demonstrator are performed. The analysis and results of their measurements are presented. In the end, some practical problems met during experiments are mentioned.
- Chapter 5: A conclusion of this thesis and proposals for future work are presented.

Chapter 2

Background

Over the last few years, the popularity of Internet has increased dramatically together with video streaming. The report from Cisco[8] figures out that Internet video traffic counts for 51% of global consumer Internet traffic in 2011. This percentage will be increased to 55% by 2016. The demand for efficient streaming technologies has become very desirable. Multicast is an efficient communication mechanism which meets this demand. This chapter gives relevant background on multicast and its applications in streaming. In addition, an overview of commercial media streaming systems and their underlying streaming protocols is presented. Motivated by these, we suggest the implementation of a global multicast demonstrator for live video streaming in a commercial media streaming system - Adobe's Flash platform.

2.1 Multicast

Multicast refers to the one-to-many or many-to-many communication. In a multicast system with n clients, the server does not need to send n identical packets. With one-to-many communication, only one identical packet is sent from the server. Each packet is replicated and forwarded on network's node such as router or client. For many-to-many communication, a client obtains some packets from other clients instead of from the server.

Multicast was first proposed by Dr. Stephen Deering in his doctoral work in late 1980's. His work proposed two protocols: IGMP (Internet Group Management Protocol) and DVMRP (Distance Vector Multicast Routing Protocol). These two protocols are multicast solution suite at the network layer[9]. Motivated by Dr. Deering's work, an experimental Multicast Backbone (MBone) was proposed. Its first practical application was implemented by IETF (Internet Engineering Task Force) in March 1992. The application sent the audio data of a meeting in San Diego to twenty sites worldwide. In the beginning, the MBone was only a virtual multicast network, in which the multicast routing function was provided by workstations. Furthermore, many protocols for IP multicast have been proposed and deployed, for both intra-domain and inter-

domain routings. At present, IP multicast has been included in the standard set of protocols shipped with most commercial routers. The commercial deployment of IP multicast has become possible. However, the deployment in the Internet is very slow due to many commercial and political problems.

To solve the deployment problem of IP multicast, it has been proposed to implement multicast at the application layer. In the beginning, the end hosts take over the functionalities of multicast-enabled routers. The end hosts replicate and forward the packets. They also manage the group membership and establish the multicast tree. This multicast is named Application Layer Multicast (ALM). Some examples using ALM are Narada[7], NICE[4] and TAG[16], etc. ALM also has its drawbacks such as unfair resource distribution and difficulties in handling frequent churn. The implementations of ALM is not very successful in real world.

Recently, peer-to-peer (P2P) technology has been adopted for live video streaming. P2P has been successfully implemented in file sharing and Voice over IP (VoIP). Both commercial and academic implementations of P2P streaming systems have achieved business success, for example, pplive, ppstream, CoolStreaming, etc. Multicast implemented at the application layer is known as P2P multicast. P2P multicast has two categories: tree-based and mesh-based. The tree-based P2P multicast is to establish multicast tree at the application layer, such as the traditional ALM. The mesh-based P2P multicast implements P2P technology similar to BitTorrent in a way.

This section presents multicast both at the network layer - Internet Protocol (IP) Multicast, and at the application layer - Application Layer Multicast and P2P Multicast.

2.1.1 Internet Protocol (IP) Multicast

Internet Protocol (IP) multicast is widely deployed in enterprises, commercial stock exchanges and multimedia content delivery networks. A common enterprise application of IP multicast is for IPTV, such as distance learning and video conferences. IP multicast implements multicast service at IP routing level, i.e. at network layer. With this service, the source only transmits one copy of individual packet. Each packet is replicated at routers and forwarded to multiple receivers simultaneously. The key concepts of IP multicast include multicast group address, multicast distribution tree and group membership management.

Multicast Address

Multicast address refers to a logical identifier for a group of end hosts. IPv4 multicast addresses are defined with the leading address bits of 1110 and classified into ″class D″ addresses, from 224.0.0.0 to 239.255.255.255. Some of these addresses are reserved for ″well-known″ purposes by the Internet Assigned

Number Authority(IANA). These reserved addresses can be classified into the following ranges:

- The address block from 224.0.0.0 to 224.0.0.255 is locally scoped and the routers will never forward.
- The address block from 224.0.1.0 to 224.0.1.255 is globally scoped and the routers will never forward.
- The address block from 239.0.0.0 to 239.25.25.255 is administratively scoped and used within the confines of an origination. They should never be seen on the Internet.

IGMP - Internet Group Message Protocol

IGMP is used by hosts and their adjacent routers in IP networks to establish multicast group membership. This protocol allows a host and its adjacent multicast-enabled routers to exchange messages. These messages describe the wishes of the host to participate or leave the multicast groups. The network architecture using IGMP is illustrated in Figure 2.1. IGMP is used both at the host side and at the router side. The operating systems at the host side shall support IGMP. Most of currently-used operating systems such as FreeBSD, Linux and Windows, support IGMP. The host sends join/leave message to the multicast-enabled router, which is connected to the other part of network. The host joins/leaves the multicast group through the local router.

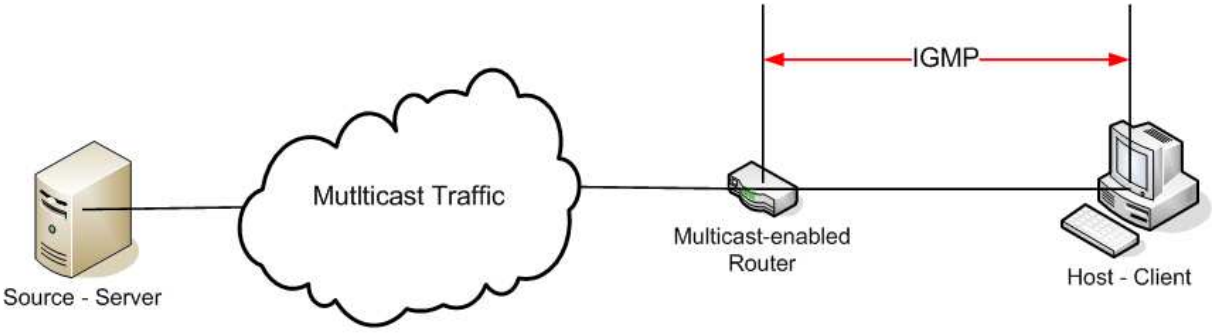


Figure 2.1: Network architecture using IGMP.

This protocol has been developed into three versions defined in "Request for Comments" (RFC) documents of Internet Engineering Task Force (IETF). All versions include the definition of join/leave procedures for a host. The host joins/leaves the multicast groups by exchanging two defined membership query messages: Membership Query and Report. The join/leave procedures can be described as below:

- **join**: A multicast-enabled router sends out periodic queries to all hosts on a given attached network using the 224.0.0.1 multicast address. If one

host is interested in the multicast traffic, it will respond to the query message and return a separate Membership Report to the router. This Report shows that this host is interested in joining the multicast group G from the specified source S. The newly discovered (S, G) will be inserted into the router's (S, G) table. This host joins the multicast group.

- **leave**: When a multicast-enabled router does not hear a report back for a multicast group after some amount of time has passed, the router then clears that (S, G) entry from its (S, G) table. This host leaves the multicast group G.

In the later two versions of IGMP, a few changes have been introduced:

- An explicit "leave" message is introduced to enable the host to leave the group without waiting.
- The qualifying multicast-enabled router with lowest IP address is selected as the one to query.
- The security feature ability is introduced and each host is allowed to select the source address which it wishes to listen to.

Multicast Routing Protocols

Multicast routing protocols are used to construct the multicast distribution trees, which consist of a collection of multicast routers in NM. There are two kinds of multipoint traffic distribution patterns: *Dense Mode* and *Sparse Mode*. The dense mode refers to that many of the nodes are interested in the multicast traffic. The dense mode multicast establishes the source-based distribution tree. The source initially broadcasts to every node. The node not interested in joining the multicast group sends a prune message to the router. After receiving this prune message, the router will not send packets to this node. There are three dense mode multicast protocols: Distance Vector Multicast Routing Protocol (DVMRP), Extends OSPF (MOSPF) and Protocol Independent Multicast - Dense Mode (PIM-DM).

The sparse mode is the opposite of the dense mode and only small percentage of the nodes wish to receive packets from the multicast group. The sparse mode multicast constructs the shared distribution tree based on a core or rendezvous point. The shared tree can be established using existing unicast routing tables. There are two sparse mode multicast protocols: Protocol Independent Multicast - Sparse Mode (PIM-SM) and Core-Based Tree (CBT).

Distance Vector Multicast Routing Protocol(DVRMP)

The DVMRP protocol is the first multicast routing protocol. DVMRP is proposed by Deering[9]. This protocol is described in RFC 1075[32] and has been

implemented in the traditional MBone. It is derived from the Routing Information Protocol (RIP) in RFC 1058[14]. DVMRP is only an experimental multicast routing protocol and it uses distance vector technique based on Bellman-Ford algorithm[9].

DVMRP performs the standard *flood-and-prune* procedure to construct a multicast distribution tree, which is called reverse shortest path tree.

- *flood*
  The router will send periodic "Hello" messages on all of its outgoing interfaces using multicast address 224.0.0.4. When receiving this message, the routers perform a reverse path forwarding (RPF) check, which check whether the incoming interfaces on the routers can reach the source in the most efficient path. The interface with successful RPF check will be added into the multicast routing table in the router.

- *prune*
  If some uninterested neighbor routers of one router become interested in joining the multicast traffic, they will send a *graft* message to the router. The router will change the prune status of these neighbor routers in its multicast routing table.

Finally, each router stores a multicast routing table, which gives the shortest path for all multicast groups.

DVMRP is a kind of distance vector routing protocols and suffers from the well-known scaling problems as the other distance vector routing protocols. This is due to that DVMRP needs flooding frequently and the flooding has its own flat unicast routing mechanism. In addition, DVMRP uses the *prune* mechanism to determine whether the packets are delivered, and to keep the state information for each source at every router. If a multicast group is densely populated with many group members, only a few neighbor routers outside the group need to send the *prune* message. However if a multicast group is not densely populated, most neighbor routers have to send the *prune* message. Each router has to store state information in MRT even for non-existing downstream group members. In this case, a significant amount of bandwidth and storage may be wasted.

Protocol Independent Multicasting - Dense Mode (PIM-DM)

PIM is a multicast routing protocol developed by IDRM (Inter-Domain Multicast Routing) working group of IETF. This protocol was proposed in order to operate under any underlying unicast routing protocol and to make use of existing routing/topology tables for RPF checks. PIM supports two different types of multipoint traffic distribution patterns: dense and sparse modes.

PIM-DM is a multicast routing protocol for dense mode and very similar to DVMRP. The two major differences from DVMRP are:

- PIM-DM can use any existing unicast routing table for RPF checks and DVMRP has to keep the multicast routing table.

- In PIM-DM, each router firstly sends the flood message to all of the outgoing interfaces. Then the neighbor routers perform RPF checks and generate prune message if RPF checks fail. In DVMRP, each router performs RPF checks for all its neighbors and only sends flood message to the neighbor routers with successful RPF checks.

Protocol Independent Multicasting - Sparse Mode (PIM-SM)

PIM-SM provides a more efficient mechanism for the sparse mode multicast traffic. It defines a "rendezvous" point (RP) between the source and the receiver for each multicast group. One multicast group has only one RP, which is aware of the source. With RP as the core router, a shared tree - Rendezvous Point Tree (RPT) can be established between the RP and the interested receivers. Now if a receiver wishes to join one multicast group, it is unnecessary to know the address of the source. The receiver only needs to notify RP by sending explicit "JOIN" message to the RP. The RP stores the receiver's routing information. The packets are forwarded from the source first to RP with unicast routing, and then to the receivers with the stored routing information via RPT. This process is illustrated in Figure2.2.

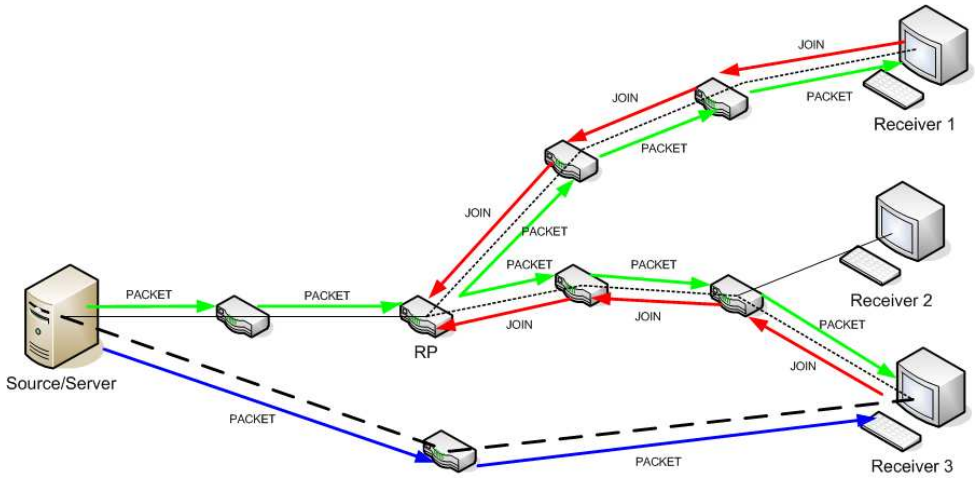


Figure 2.2: PIM-SM process: The interested receivers 1 and 3 send explicit "JOIN" message to the RP via RPT. The packets are sent firstly to the RP and then to the receivers 1 and 3. After a while, the data transmission between the source and the receiver 3 can use the shortest path along the blue arrows.

In addition, PIM-SM provides another option by shifting to the native data transmission via SPT. The RP will quit the data transmission process by sending a Register-Stop message to the source. This mechanism can make sure efficient transmission between the source and the receiver. It can also avoid heavy traffic on the RP.

PIM-SM is a widely used sparse mode protocol and has some advantages compared to other two dense mode protocols. Sparse mode protocol provides better scalability and only routers on the path between the source and the receiver need to keep membership state. Secondly, sparse mode protocol only forwards packets to the router which has sent explicit "JOIN" message. However, the sparse mode protocol depends heavily on the core router a lot and therefore the core router can be a single point of failure.

Limitations of IP multicast

IP multicast is efficient and can result in superior bandwidth utilization. It once attracted significant attention and was proposed as a promising solution for live video streaming. However, its deployment remains limited due to many practical issues. *Diot et al.*[10] present an extensive overview of problems in current implementation and deployment of IP multicast. According to [10], current IP multicast lacks simple and scalable mechanisms for supporting access control, security, address allocation and network management. They come to the conclusion that IP multicast does not satisfy the requirements of Internet Service Provider (ISP). IP multicast might be deployed inside an Autonomous System (AS), but not widely adopted outside.

In addition, IP multicast increases the router complexity. Each router has to maintain its associated group members and exchange this information with other routers. This violates of the stateless principle of routers and the best-effort mechanism of IP network.

In conclusion, IP multicast is not appropriate for global video streaming. But it can be an efficient solution for video streaming services provided by a single ISP.

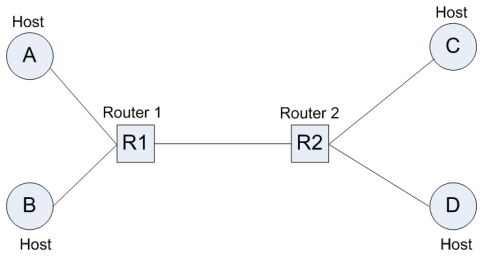
2.1.2 Application Layer Multicast (ALM)

The basic idea of ALM is to replicate data packets at end hosts instead of routers as IP multicast does. The multicast functionality is moved from the IP network layer to the application layer. The participating end hosts constitute the multicast tree. The unicast tunneling mechanisms are used to deliver data between two end hosts. A comparison of Unicast, IP Multicast and Application Layer Multicast is shown in Figure 2.3.

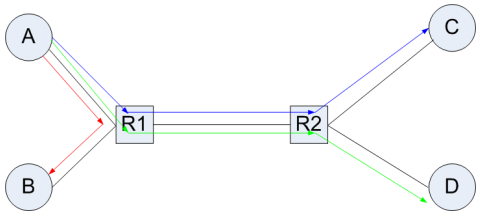
There are three different categories of ALM techniques: mesh-first, tree-first and implicit approaches. Three ALM approaches *Narada*[7], TAG[16] and NICE[4] are described in this section.

Narada

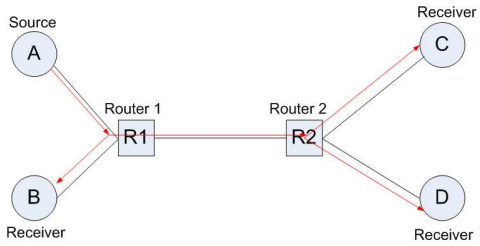
In [7], *Chu et al.* propose End System Multicast (ESM) and present the protocol *Narada* ESM is one of first efficient multicast overlays. The protocol *Narada*



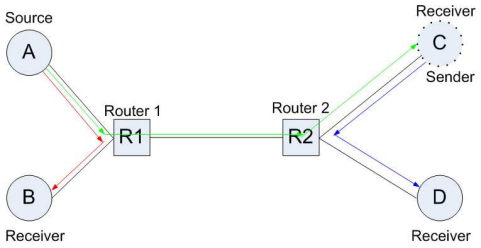
(a) Example: the two squares present the routers and the four circles present the end hosts.



(b) Unicast: three copies of packets are sent from the source A to three other hosts.



(c) IP Multicast: only one copy of packet is sent from the source A, and packets are replicated in the routers and forwarded to other hosts.



(d) Application-Layer Multicast: two copies of packets are sent from the source A, and packets are replicated in the host C and forwarded to the host D. The host C is both a receiver and a sender.

Figure 2.3: Comparison of Unicast, IP Multicast and Application-Layer Multicast.

can achieve all multicast related functionality such as membership management and packet replication. This protocol constructs an overlay spanning trees in a two-step process.

1. Construction of a *mesh*.
   The distributed algorithms are used to construct and maintain the mesh. For each member, a refresh message is generated periodically and exchanged between neighbors. The refresh message contains a list of entries, one entry for every other member in the same group. The refresh message is used for group management, such as the maintenance of the mesh, the join of new members, the leave and failure of members, etc.

2. Construction of the spanning trees based on the mesh.
   The optimal trees for the individual source are constructed using existing routing algorithms, such as a distance vector protocol. Each tree is rooted at one source. The tree is constructed using the reverse shortest path as in DVMRP.

According to the simulations and Internet experiments in [7], it is concluded that End System Multicast using *Narada* can achieve good performance for small and medium sized groups involving tens to hundreds of members. For larger size groups, ESM architecture can not keep such good performance. The reason is that *Narada* requires each member to keep information

about all other group members. Increased number of group members results in more memory for storing such information and adds a lot of control overheads.

TAG - Topology-Aware Grouping ALM

In [16] *Kwon et al.* investigated a heuristic application-layer multicast approach - Topology Aware Grouping (TAG). TAG uses the core-based multicast protocol. TAG constructs efficient overlay networks by exploiting underlying network topology information. The path information maintained in the routers are used to determine the shortest path to the given root. When a new member joins into one group, this member determines its best parent and children based on the existing path information. There are no needs for extra computations to construct the multicast tree for this new member. When a member leaves the group, it sends a explicit LEAVE message to its parent. In the children list of its parent, the leaving member is removed and its children are inserted. In TAG, each node maintains only the information of its parent and children nodes.

NICE

In [4], *Banerjee et al.* presents an application-layer multicast protocol - NICE. The protocol in NICE is specially designed for the applications with very large size group and low bandwidth real-time data. The applications include news and sports ticker services, real-time stock quotes and updates and popular Internet Radio sites.

The goal of NICE protocol is to develop an efficient, scalable, and distributed tree-building protocol which does not require any underlying topology information. A hierarchically connected control topology is created by assigning members to different layers from L_0 to L_n. All hosts are included in the lowest layer L_0. In the highest layer L_n, there is only a single host. The hosts in each layer are grouped into a set of clusters. In each cluster, the host with the minimum distance to all other hosts in the cluster is selected as the leader of this cluster. All cluster leaders in the same layer L_i constitute the upper layer L_{i+1}. The clusters and layers are created using a distributed algorithm described in [4]. The requirements for the distribution of hosts in the different layers are listed:

- A host belongs to only a single cluster at any layer.
- If a host is present in some cluster in one layer, this host must be included in at one cluster in all of its lower layers.
- If a host is not present in one layer, this host cannot be present in any of its higher layers.

- Each cluster has its size bounded between k and $3k-1$, where k is a constant.

- The number of layers is determined by the size of the cluster and the number of hosts.

NICE uses different overlay structures for control messages and data delivery respectively. In the control topology, each host in a cluster exchanges the cluster membership information with other hosts. The source-specific tree for data delivery is constructed based on the control topology.

NICE achieves the same performance as *Narada* with some improvements, such as lower control overhead. Moreover it has better performance for large size group than *Narada*.

Comparison of IP multicast and ALM

Compared to IP multicast, ALM has the following advantages:

- The best-effort mechanism can be maintained using ALM. No significant changes are needed in existing IP networks.

- In ALM, it's possible to support some higher layer functionalities, such as error, flow and congestion control, etc. These functionalities are difficult to be implemented in IP multicast.

- The inter-domain multicast in ALM can use the existing inter-domain protocols. The particular inter-domain protocols should be used in IP multicast.

ALM has the following disadvantages compared to IP multicast:

- ALM is efficient compared to the unicast. However it cannot achieve the same efficiency as IP multicast does. The control overheads and duplicated packets are delivered at the network layer.

- The latency is increased because communication between hosts involves traversing other hosts. If the replication of packets happens on one host, other hosts have to wait until the packets have arrived. Extra delay is introduced for other hosts.

- All hosts have to handle all multicast related functionality, such as group managements and data replications.

In conclusion, ALM is applicable to global streaming services on current unicast architecture. It is not as efficient as IP multicast, but more efficient than unicast.

2.1.3 Peer-to-Peer (P2P) Multicast

A P2P system enables the sharing of data and resources located on all end hosts throughout a network. Such a system allows each end host to contribute resources to other end hosts and requires resources from other end hosts. An end host in a P2P system is named as a peer or a node. All peers are connected using virtual links on the overlay network and form an overlay topology. It fits the definition of ALM in which the end hosts form a multicast tree on the overlay network. The use of P2P technology has achieved tremendous success in the applications for file-sharing and Voice over IP (VoIP). Recently, the P2P technology has been proposed and deployed into live video streaming. The P2P-based live video streaming is a subset of ALM systems and named as P2P multicast. Existing approaches in P2P multicast can be classified into two categories: tree-based and mesh-based according to their overlay topology[19].

Tree-based P2P multicast

Tree-based P2P multicast closely resembles the design of IP multicast. This approach is to construct a multicast tree on the overlay network. The tree is originating from the streaming source. The nodes of the tree are peers. The peer joins the tree at certain level. It receives the packets from its parent peer and forwards the packets to its children peers. The delivery of packets is pushed down the tree from the source. Each peer just forwards every packet it receives to its children peers. It does not schedule packets it receives. It does not query its children peers whether they have received the forwarded packet. This delivery mechanism is referred to as push-based transmission. This mechanism provides low latency and in-order delivery of packets. Tree-based P2P multicast has developed from single-tree based approach to multi-tree based approach.

ESM[7] introduced in Section 2.1.2 is one of single-tree based approaches. Such approach just constructs a single multicast tree at the application layer. One example of tree is illustrated in Figure 2.4. The single-tree based approach suffers two major drawbacks. One is that all leaf peers don't forward the received packets. This implies that the leaf peers don't contribute their upload bandwidth. A large-size P2P system contains a large portion of peers as the leaf peers. This greatly degrades the bandwidth utilization and causes unfair contributions among all peers. The high-level peers have more loads and are more complicated than the leaf peers. Another drawback is that the departure or failure of high-level peers can cause significant program disruption and requires the re-construction of the overlay topology. It has been shown that the recovery is not fast enough to handle frequent peer churn.

To cope with the drawbacks in single-tree based approach, the multi-tree approach has been proposed[5]. It constructs multiple sub-trees and splits the stream source into multiple sub-streams. Each sub-stream is delivered over a sub-tree. One example with two sub-trees and sub-streams is illustrated

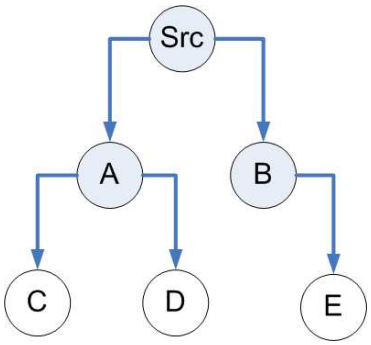


Figure 2.4: Single-tree overlay topology: The circles A, B, C, D and E are five peers. The circle with "Src" represents the streaming source. The packets are sent from the source to all nodes in this tree. The circles C, D, E are leaf peers.

in Figure 2.5. Such approach addresses the leaf peers problems in single-tree based approach. A peer has different position in different sub-tree. It might be a leaf peer in one sub-tree and might not be a leaf peer in another sub-tree. It provides better resilience against peer churn and fairer bandwidth utilization from all peers. However, it introduces the complexity in the other aspects. For example, it requires special multi-rate or/and multilayer encoding algorithms for splitting stream. It also requires disjoint multicast trees, which can be difficult in the presence of network dynamics. Therefore, the multi-tree based P2P multicast has not been demonstrated feasibly in the real system over the Internet.

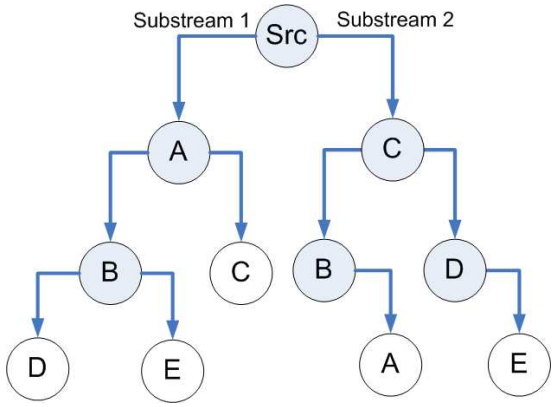


Figure 2.5: Multi-tree overlay topology: The circles A, B, C, D and E are five peers. The circle with "Src" represents the streaming source. The source is divided into two sub-streams and two multicast trees are constructed. The packets from each sub-stream are sent from the source to all peers in each tree. The circles without filling represent leaf peers.

Mesh-based P2P multicast

Mesh-based P2P multicast is motivated from the success of file swarming mechanism, such as BitTorrent. This approach is to form and maintain a

mesh topology among peers. One snapshot of a mesh overlay is illustrated in Figure 2.6.

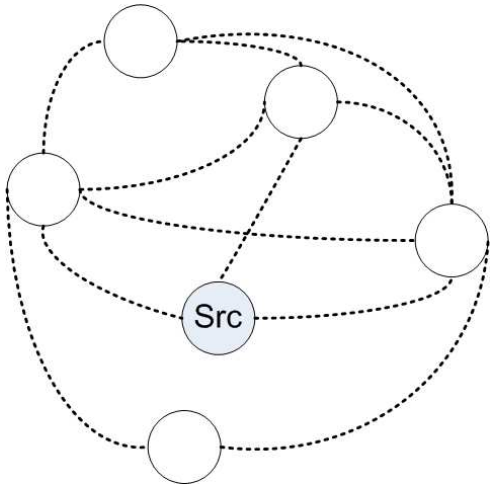


Figure 2.6: A snapshot of a mesh overlay: The circles without filling are five peers. The circle with "Src" represents the streaming source. The dotted lines represent the peering connections. The connections are formed and discarded over time. This mesh is not static and changes over time.

In a mesh overlay, each peer connects to multiple neighboring peers. A peering connection is established based on the mutual agreement between two peers. Once the peering connection is formed, two peers exchange keep-alive messages regularly. If a peer does not receive keep-alive message from another peer within a pre-configured timeout period, this indicates that another peer leaves or fails. The peer will find new neighbors to keep the desired level of connectivity, such as the number of neighbors. The mesh overlay is not static and changes over time.

A peer may receive packets from its neighboring peers and forward packets to these peers simultaneously. There are two major ways of data exchange between the peer and its neighbors: push and pull. With the push approach, a peer actively forwards a packet it receives to its neighbors. The peer does not check whether its neighbors have this packet. Therefore, a peer could receive the same packet from its neighbors. This results in a waste uploading bandwidth in redundant pushes. The pull approach can avoid redundant pushed. With this approach, a peer and its neighbors periodically exchange information about their packet availability using buffer maps. After the peer obtains the buffer maps from its neighbors, it can find which packets it lacks can be found in one of its neighbors. The peer will send a request to the neighbor and pull the missing packets from it. However, the pull approach introduces more signaling overhead and additional latency. A peer and its neighbors have to exchange buffer map frequently. The pulling requests are introduced in addition to video data. Moreover, the pulling of a video packet has to wait at least one round delay until the pulling request is responded.

It is also possible to combine push and pull data exchange approaches. This approach is named as hybrid push-pull approach.

There are many recent P2P streaming systems that adopt mesh-based approaches[20][22][34]. *Magharei et al.* in [21] compare the performance of tree-based and mesh-based approaches using simulations. They come to the conclusion that the mesh-based approach has a superior performance over the tree-based approach. This result is reasonable mainly based on the following points. Firstly, a streaming system with mesh can has better ability to cope with churn. A peer has two or more neighbors. If a peer's neighbor leaves, the peer can still receive packets from remaining neighbors. It does not need to immediately reconstruct the overlay as in multicast trees. Secondly, each peer in such systems has similar functionalities. All peers can make use of their uploading bandwidth and obtain fair bandwidth utilization. Finally, the mesh topology is dynamic. The peering connections are formed and discarded over time. If peers depart or fail, new peering connections will be established among remaining peers. If new peers join, they will be connected to old peers.

2.2 Media Streaming Systems

A media streaming system refers to a set of applications in that a provider constantly delivers media data to the end hosts. At the same time, the media data is played immediately at the end hosts. A typical media streaming is illustrated in Figure 2.7. The media data can be stored files or captured directly from camera. The stream data is encoded for transmission, storage and encryption. The encoded stream is decoded for playback at the end host. The device or software for encoding and decoding media is a codec. The media server is application software that can distribute the media streams to many end hosts. The hosts have client applications that receive and play the media streams. The hosts can be desktops, connected TV, tablets, and smart phones, etc. In addition, an important technology in the media streaming system is the streaming protocol. The streaming protocol defines how to deliver media data over Internet and how to control the delivery to obtain smooth playback.

2.2.1 Media Servers

A media server is application software that distributes the media streams. The software usually is located in a computer with server OS. It is also called an application server. The popular media servers in current market include Microsoft's IIS (Internet Information Services) Media Server, Adobe's Flash Media Server and Wowza. They are presented in this section.

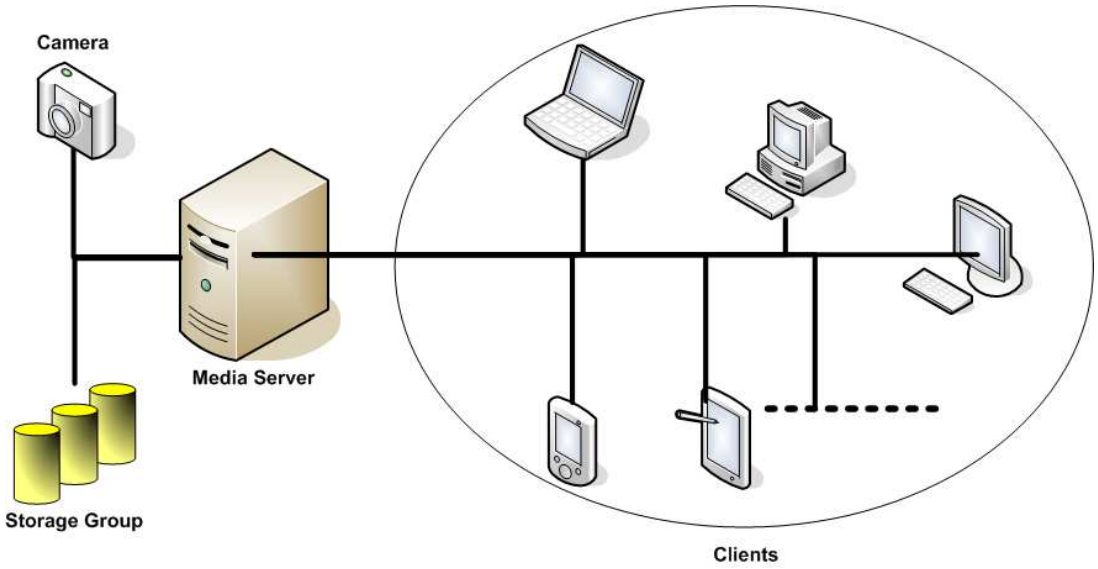


Figure 2.7: Illustration of a typical media streaming system.

IIS Media Services

IIS Media Services are media streaming service packages for Microsoft's Windows Servers. They include a web server application and set of feature extension modules. The modules provide an integrated HTTP-based media delivery platform. The packages support Smooth Streaming protocol for streaming over HTTP. They also support Microsoft's Silverlight and Apple's iOS at client sides. In addition, Microsoft provides Software Development Kit (SDK) that allows developers to create their own applications. The newest released version - IIS 7.5 is included in Windows 7 and Windows Server 2008 *R*2.

Flash Media Server (FMS)

Flash Media Server is an application server from Adobe. It supports multiple streaming protocols, including Real-Time Message Protocol (RTMP), Real-Time Media Flow Protocol (RTMFP), Adobe HTTP Dynamic Streaming and Apple HTTP Live Streaming, etc. Its major supported client applications are based on Flash Player. Adobe also provides APIs based on ActionScript Language. These APIs can be used by the developers to create their own applications. The newest version of Flash Media Server is Adobe Media Server 5. It has four editions with different supported features: Adobe Media Server Standard, Adobe Media Server Professional, Adobe Media Server Extended and Adobe Media Server Development Starter (free). In this thesis the earlier version - Flash Media Server 4.5 is used. It has also four editions with different names: Adobe Flash Media Streaming Server, Adobe Flash Media Interactive Server, Adobe Flash Media Enterprise Server, and Adobe Flash Media Development Server (free). Flash Media Server can work on Microsoft Windows Servers and Linux Servers.

Wowza Media Server

Wowza Media Server covers the most broad range of streaming protocols and clients technologies with lower price. It is a tightly 64−bit Java server and requires Java Runtime Environment (JRE) or Java Development Kit (JDK). Due to the portability of Java, Wowza Media Server is supported by almost all available server OS. The newest version is Wow Media Server 3.

2.2.2 Streaming Protocols

A streaming protocol defines how to deliver the media data over Internet. It is usually on top of transport layer protocols. The main tasks of streaming protocols are to pack the media data into packets for transmission, to control the delivery of packets for a continuous playback, etc. This section introduces several standard streaming protocols: Real Time Transport Protocol (RTP), Session Initiation Protocol (SIP) and Dynamic Adaptive Streaming over HTTP (DASH). These standard streaming protocols have been widely implemented in current commercial streaming systems. The standardization of streaming protocols is important for business success of streaming services. As a pioneer in the field of streaming service, Adobe has designed and implemented steaming protocols for its Flash platform: Real Time Messaging Protocol (RTMP) and Real Time Media Flow Protocol (RTMFP). The two protocols from Adobe are also presented in this section.

Real Time Transport Protocol (RTP)

The RTP is a standardized transport protocol for real time application defined in RFC 3550[25]. It defines a standardized packet structure for delivering the media content. It appends header fields to the audio/video chunks before the sender passing them to the transport layer. These header fields include the payload type, the sequence number, the timestamp, the synchronization source identifier and some miscellaneous fields. An RTP header is illustrated in Figure 2.8. The RTP header appending an audio/video chunk forms an RTP packet. The RTP packet is then encapsulated into a UDP packet for delivery over IP network.

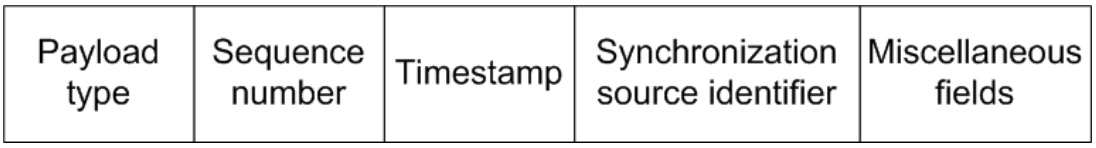


Figure 2.8: RTP header fields

The RTP only provides an encapsulation mechanism, but without any mechanism to ensure timely delivery of data or other quality-of-service (QoS) guarantees. In addition, RFC 3550[25] also specifies RTP Control Protocol (RTCP)

as RTP's companion protocol. This protocol defines the RTCP packets. These packets do not encapsulate any media content. These packets are sent periodically and contain the reports that announce statistics about the sender and/or the receiver. These statistics include the number of sent packets, the number of lost packets, and the inter-arrival jitter. These statistics can be used by the application developer for different purposes such as monitoring and modifying the transmission rates, diagnosing the network problems, etc.

The RTP is often used in conjunction with the Internet telephony standards, such as Session Initiation Protocol (SIP) and H.323.

Session Initiation Protocol (SIP)

SIP is a standard application-layer signaling protocol defined in RFC 3261[24]. RFC 3261[24] specifies the SIP messages for establishing, managing and terminating the sessions between two participant over the Internet. The SIP applications include Internet telephone calls, multimedia distribution, and multimedia conferences, etc. The SIP messages do not include the media data and are only used for creating the sessions. The media data transmission usually uses other transport protocols such as RTP. The messages are ASCII-readable and resemble HTTP messages, such as INVITE message, 200 OK message, ACK message, REGISTER message and BYE message etc. All messages require to be acknowledged. The SIP protocol is independent of the underlying transport layer. It can run on top of UDP or TCP. A basic procedure to establish sessions between two SIP clients A and B is illustrated Figure 2.9.

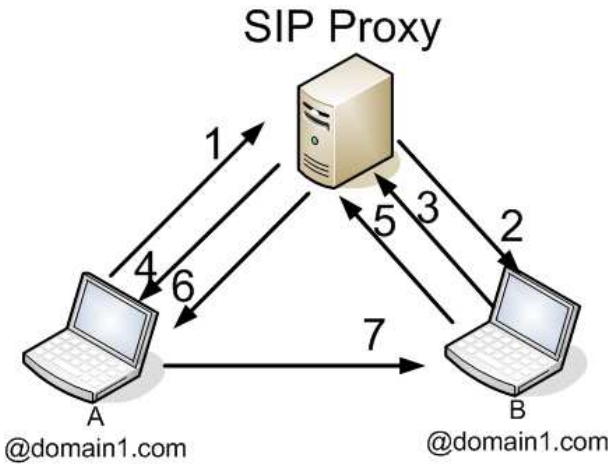


Figure 2.9: SIP basic establishment procedure: the system consists of one SIP proxy and two SIP clients - A and B. The address of SIP clients are e-mail-like address. These steps are taken: (1)A sends an INVITE message to a SIP proxy. (2)The SIP proxy relays the INVITE message to B. (3)B sends a 180(ringing) message to the proxy. (4)The proxy relays the 180(ringing) message to A. (5)B sends a 200 OK message to proxy. (6)The proxy relays the 200 OK message to A. (7)A sends an ACK message directly to B. Now A knows the contact address of B via the 200 OK message. A session establishment is completed between A and B.

Dynamic Adaptive Streaming over HTTP (DASH)

Currently, it has become practical to deliver the media content in large segments using HTTP. HTTP streaming has become a popular approach in commercial streaming services, such as Microsoft's Smooth Streaming, Apple's HTTP Live Streaming and Adobe's HTTP Dynamic Streaming. The HTTP is used as the underlying data delivery method. Different manifest and segment formats are used for different streaming server and clients. MPEG has recently developed a standard for HTTP streaming. This standard is known as MPEG DASH and is expected to be published as ISO/IEC 23009-1[26].

A simple case of DASH is illustrated in Figure 2.10.

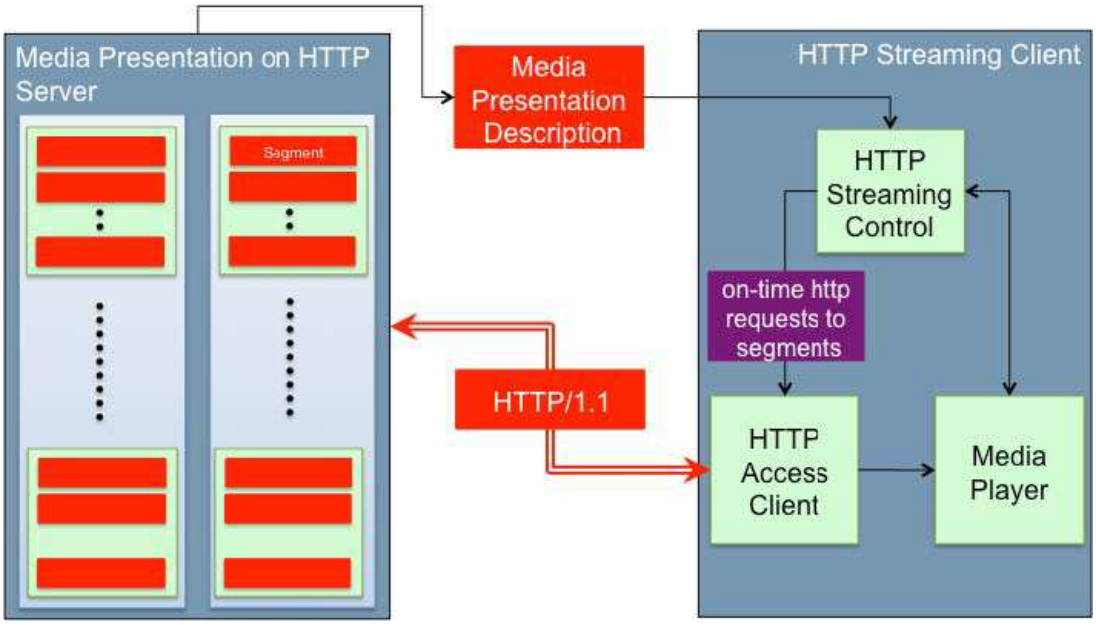


Figure 2.10: A simple case of DASH.[28]

The media content is captured and stored on an HTTP server in the form of Media Presentation. A Media Presentation is a structured collection of encoded data of the media content. It consists of a sequence of one or more Periods as shown in Figure 2.11. The media content is divided into the consecutive and non-overlapping Periods according to the time. A Period contains one or more Representations from the same media content. The Representations differ by the encoding choice, for example by bit-rate, resolution, language, or codec. Each Representation consists of one or more Segments. The Segment is the actual media bit-streams in the form of chunks. It can be uniquely referenced by an HTTP-URL.

In addition to the media data, Media Presentation Description (MPD) is used to describe Media Presentation. The MPD is a manifest file which describes available media content, various alternatives, URL addresses and other characteristics. The MPD is usually in the form of XML document. The HTTP streaming client first obtains the MPD. The MPD can be delivered in many

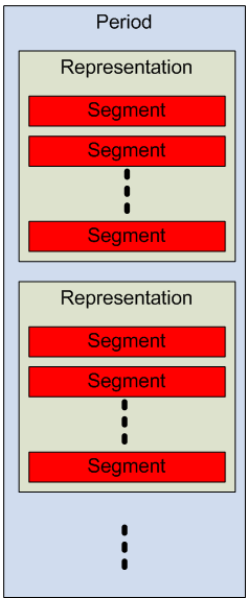


Figure 2.11: A Period of a Media Presentation.

manners, including using HTTP, email and broadcast, etc. The MPD may be updated during the streaming.

Once the client obtains the MDP, an HTTP streaming control first parses the MDP. The control will select the set of Representations based on the MDP, the client's capabilities, and user's choices. According to its selection, the control sends on-time HTTP requests to the HTTP access client. The access client downloads the media segments using HTTP GET or partial GET methods. The downloaded segments are sent to the media player for playing.

The DASH solution provides an industry-standard for media streaming over Internet with many benefits. Firstly the DASH can easily avoid problems with firewall and NAT traversal. Such problems happen typically in the streaming services using RTP/UDP. The DASH provides the ability to use standard HTTP servers and standard HTTP caches. The most of existing servers and network caches can be reused. Furthermore, it can launch streaming services on the existing successful HTTP-based Content Distribution Networks (CDNs). The DASH moves the complexity to the client from the streaming server. The client chooses the content rate to match its available bandwidth without negotiation with the streaming server.

Real Time Messaging Protocol (RTMP)

The RTMP was initially a proprietary streaming protocol over TCP developed by Adobe. Adobe has released its incomplete specification in [23]. The RTMP is an application-level protocol for multiplexing and packetizing media transport streams over TCP. It establishes an RTMP connection between the client and the server. The connection intends to carry parallel streams of video, audio,

and data messages, with associated timing information.

An RTMP connection is established through a handshake procedure. In this procedure, three static-size chunks are exchanged between the client and the server. After handshaking, the connection multiplexes one or more chunk streams. Each chunk stream carries messages of one type from one message stream. Each chunk stream is assigned with a unique chunk stream ID. These chunk streams are among a chunk stream for video, a chunk stream for audio, a chunk stream for out-of-band control messages, etc. All chunk streams are delivered independently.

A chunk is a basic RTMP packet before it is transmitted over TCP. A chunk consists of chunk header and chunk payload as illustrated in Figure 2.12. During transmitting, the sender packetizes the messages into chunks (chunking). The maximum chunk size can be dynamically negotiated between server and client. The receiver will assemble received chunks into the original messages. The chunking can split large messages into smaller chunks. This can prevent large low-priority messages from blocking smaller high-priority messages. For example, it can prevent large video data from blocking small audio data or control messages that have higher priority. The chunking can also reduce the overheads of sending small messages. The compressed representation of chunk information is included in the chunk header.

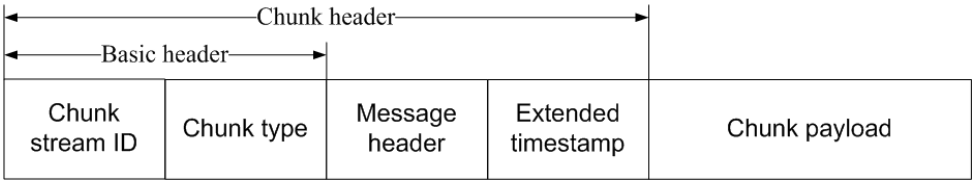


Figure 2.12: An RTMP packet consists of chunk header and chunk payload.

The chunk header contains chunk stream ID, chunk type, message header and extended timestamp. The chunk stream ID is variable-sized. The 2-bit chunk type determines the format of message header. There are four different formats of message header. The message header has variable-size among 0, 3, 7, 11 bytes. It might contain these fields: message stream ID, message type, message length, absolute timestamp and timestamp delta. The message types include audio messages, video messages, command messages, shared object messages, data messages, protocol control messages and user control messages. The extended timestamp field is used when the timestamp in the message header doesn't fit in the 24-bit fields. In some cases, some packets over the RTMP connection have no chunk header. They share the header information with their previous RTMP packets.

In addition, Adobe also introduces multiple variations for RTMP in order to increase its security and compatibility. There are two methods to encrypt RTMP sessions, RTMPS using industry standard SSL mechanisms and RTMPE using Adobe's own security mechanism. RTMPT is encapsulated

within HTTP requests to traverse firewalls. It communicates over port 80 and passes the AMF data inside HTTP POST requests and responses.

Real Time Media Flow Protocol (RTMFP)

The RTMFP is a proprietary streaming protocol over UDP on Adobe's Flash platform. It is designed and written based on Secure Media Flow Protocol (MFP) from Amicima. It provides a new transport protocol to securely deliver media flows over Internet. The protocol establishes a session between one pair of peers. The session is a bidirectional channel in which several flows travel as illustrated in Figure 2.13. A flow is a unidirectional communication channel for transporting a correlated series of user messages. The messages can be streams of audio/video, acknowledgements and other messages. In RTMFP, the combined control and media data are delivered via one channel.

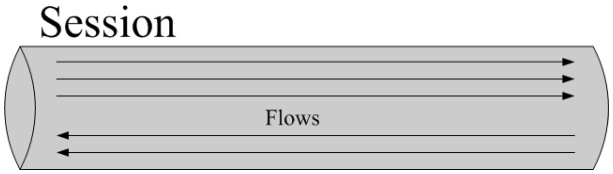


Figure 2.13: Session and flows for RTMFP: many flows are transported inside one session between a pair of endpoints.

Recently, the author of RTMFP - Michael Thornburgh posts the specification for the RTMFP base transport protocol in [31]. A typical RTMFP packet is illustrated in Figure 2.14. An RTMFP packet consists of a scrambled session ID and an encrypted packet before it is transported over UDP. The scrambled session ID is the session ID modified by performing a bitwise exclusive-or with the first two 32-bit words of the encrypted packet. The session ID identifies the session to which the packet belongs. It provides the decryption key used to decrypt the encrypted packet. The scrambled session ID looks more like noise. It can avoid NAT false-positives and annoy Deep Packet Inspection (DPI) boxes. The RTMFP packets are associated with the session ID, not with the IP address. So the RTMFP has IP address mobility. For example, an RTMFP session is up between two peers and the IP address of one peer has changed. Another peer can find the new IP address of the peer and re-establish the session. The data delivery is continued without changing the session ID in the RTMFP packets.

Figure 2.14: An RTMFP packet consists of a scrambled session ID and a encrypted packet.

The encrypted packet is a packet encrypted according to a Cryptography Profile used in Adobe Flash platform. The packet encryption uses a block cipher operating. The encrypted packets are decipherable without inter-packet dependency, since packets may be lost, duplicated, or reordered in the network. An unencrypted and plain packet consists of a variable-sized header, zero or more chunks and padding as illustrated in Figure 2.15. The header contains flags, timestamp, timestamp echo, etc. A chunk is the data unit of the RTMFP packet that consists of chunk type, chunk length and chunk payload as illustrated in Figure 2.16. The chunk type indicates the content of payload. The chunk can be used for session startup, in-session control and in-session flows. The padding is inserted to meet cipher block size constraints by the sender.

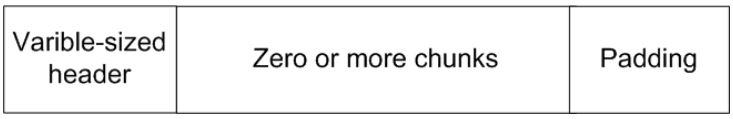


Figure 2.15: A unencrypted and plain packet consists of a variable-sized header, zero or more chunks and padding.

Figure 2.16: A chunk of the RTMFP packet consists of chunk type, chunk length and chunk payload.

An RTMFP session is established with a 4-way handshake in two round trips as illustrated in Figure 2.17. After the session is established, the initiator and the responder creates new sessions respectively with the initiator and responder session ID. The packets with the corresponding session ID are delivered between them. In addition, the RTMFP also provides other mechanisms for session establishment, such as the forwarder for NAT traversal, the redirector to supply the alternative IP addresses.

Moreover, the RTMFP adds the congestion control and avoidance algorithm according to RFC 2914[11] and RFC 5681[3]. The congestion control used is TCP compatible. There are two types of acknowledgments for user data of a flow: bitmap acknowledgement and range acknowledgement. The acknowledgements contain the sequence numbers of User Data segments that have been received. The acknowledgements can notify the packet loss and control the buffer on a receiver.

The RTMFP has integrated the security feature into the protocol itself, including the introduction of the explicit session ID and the encapsulated packet. The use of the session ID instead of IP address supports IP address mobility and prevents hijack.

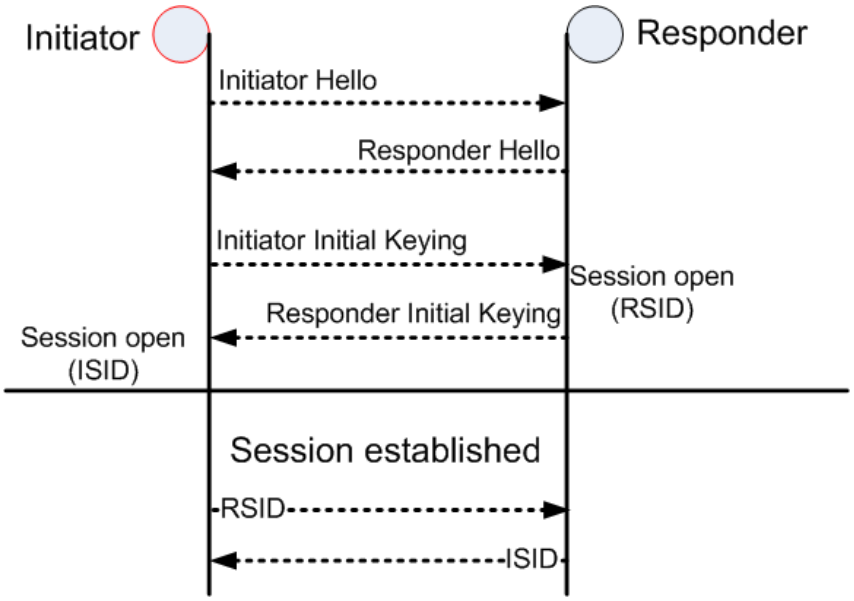


Figure 2.17: A normal session establishment for RTMFP.

2.2.3 P2P Streaming Systems

As an efficient communication mechanism, P2P multicast has been applied in the P2P streaming systems in real world. This section presents some representative P2P streaming systems, specially P2P from Adobe. The P2P on the Flash platform motivated the design and implementation of the global multicast demonstrator in this thesis.

Coolstreaming/DONet

Coolstreaming[34] is a data-driven overlay network (DONet) for live media streaming. Each DONet node periodically exchanges information about data availability and partnerships with a set of partners. The node retrieves unavailable data over connections from one or more partners. Both the partnership and the data transmission are bidirectional and flexible.

Each node maintains a membership cache (mCache) that contains a list of active nodes. The nodes in this list do not include all active nodes. The system employs the Scalable Gossip Membership protocol, SCAM[12] to provide a group membership service based on gossiping. Each node periodically generates a membership message to declare its existence. The message contains the node id and number of partners. The nodes which receive message will update the information stored in mCache. In addition, each node also periodically select random nodes from mCache and establish partnership connections with them. This results in a stable number of partners and helps to explore partners of better quality.

Each node also periodically exchanges a Buffer Map (BM) with its partners. The BM represents data availability in the buffer of a node. In the BM, each

bit is for a packet. The bit 1 indicates that the packet is available and the bit 0 otherwise. Afterwards, the node retrieves unavailable packet from its partners. The selection of fetching the expected packets from the partners uses a heuristic scheduling algorithm. For each unavailable packet, this algorithm selects the partner with the highest bandwidth and enough available time. The node will fetch the packet from the selected partner.

An experiment system of DONet is implemented in the PlanetLab[1] environment. *Zhang et al.* evaluate the performance of DONet under stable and dynamic environments. The results show that this system has low control overhead, less than 2% of the total traffic even with over $5-6$ partners. And the control overhead for each node is independent of the overlay size. The playback continuity improves with increasing number of partners. However, the improvement is very limited with more than 4 partners. In addition, the larger overlay will leads to better playback continuity due to more cooperation among nodes.

A public Internet-based DONet implementation was released under the name *Coolstreaming v.0.9* in 2004. It attracted a remarkable amount of clients, over 1 million. It is implemented using Python. The system can be used under Windows, Linux or other operating systems supporting Python. However, the Coolstreaming service was stopped due to copyright issues in 2005. Now it is the base technology for Roxbeam Inc.[2], which has launched commercial streaming services in multiple countries.

In 2008, a new version of Coolstreaming is presented in [18] with several modifications. The new system discomposes a video stream into several sub-streams by grouping video blocks. Each node in the new system maintains two buffers: synchronization buffer for each corresponding sub-stream and cache buffer that combines sub-streams to one stream. The data transmission in the new system adopts a hybrid push and pull scheme.

P2P on the Flash Platform

Adobe introduces a P2P technology on the Flash platform based on RTMFP: RTMFP Group. The RTMFP group defines a mesh overlay network for live media streaming and instant messages. The overlay topology is a ring as illustrated in Figure 2.18. Each node in the ring is represented with a given peer ID. The peer ID is determined by a SHA256 hash of cryptographic public key and other stuff. It is a 256-bit numeric identifier presented as a 64-digit hex strings. This ID cannot be directly chosen or influenced.

Each peer has a list of "heard" peers in the ring. The peer has a topology management algorithm that is repeated forever. The algorithm sorts the "heard" peers into the "desired" peers and the "undesired" peers. The peer is connected to the "desired" peers. The "desired" peers are continuously opti-

[1] http://www.planet-lab.org
[2] http://www.roxbeam.com

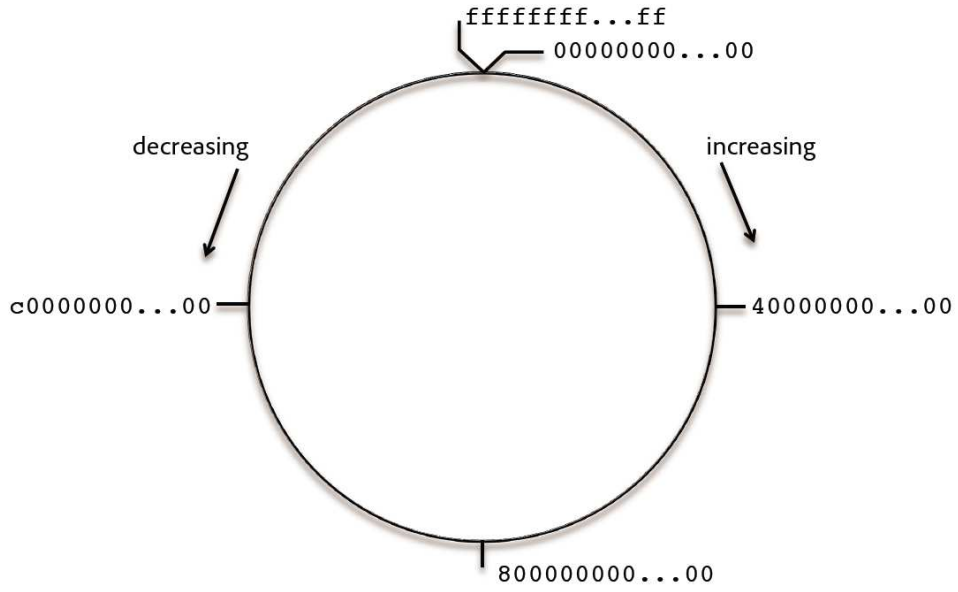


Figure 2.18: RTMFP group overlay topology: a ring mod 2^{256}.(from [30])

mized through gossip with the rate higher than natural churn. This results in a self-organizing topology. The sorting algorithm chooses the "desired" peers from the "heard" peers with the following rules. The "desired" peers are chosen from 6 closest peers in numeric space, 6 peers with measured lowest RTT and 1 peer that is picked up at random every round. This results in a fully connected ring.

Each peer contains a buffer with the video blocks as illustrated in Figure 2.19. The video blocks are listed in terms of time. The blocks are positioned into two areas: Relay Margin and Window Duration. The peer only attempts to get every block in the Window Duration. If any missed block is outside the Window Duration, the peer no longer attempts to get it. The Relay Margin provides extra time for in-flight requests to be fulfilled.

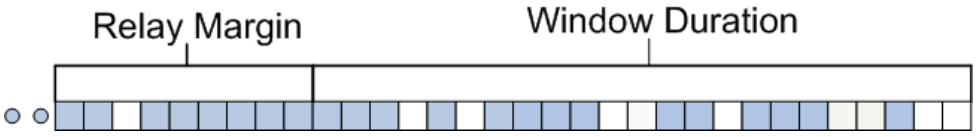


Figure 2.19: Map of video blocks in one peer for P2P on the Flash platform. It is divided into two areas: Relay Margin and Window Duration. Blocks without filling represent current missed ones.

The data delivery between two peers is a hybrid pull-push approach. The data exchanging between two peers are video blocks inside the Window Duration. The pull approach is illustrated in Figure 2.20. In this example, a peer sends a bit map for the video blocks inside the Window Duration to all neighbor peers. A bit with 1 represents the available block in the peer and a bit

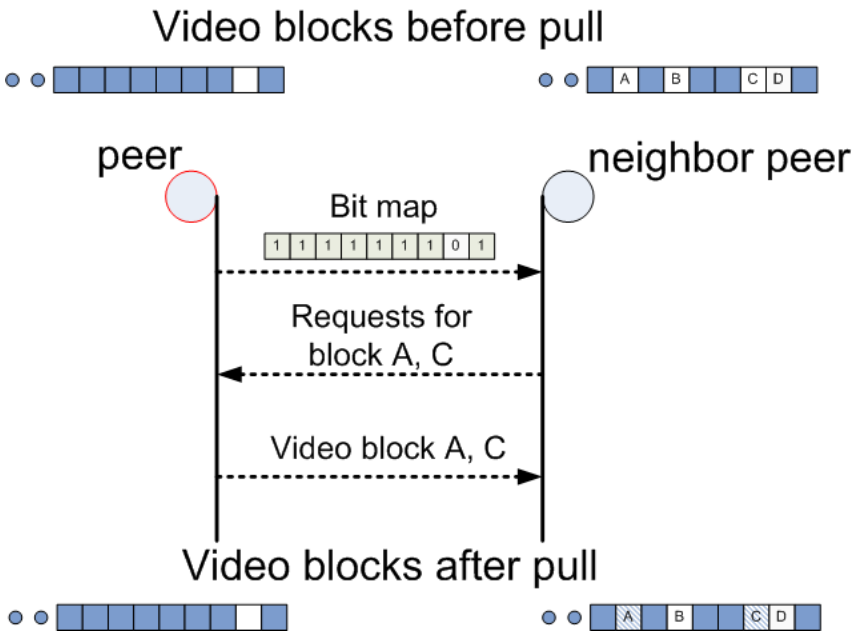


Figure 2.20: The pull approach for P2P on the Flash platform.

with 0 for the missed block. A neighbor peer that has received the bit map will pick some of its missed blocks from the original peer and sends the requests for them. The peer responds to these requests and sends the required video blocks to the neighbor peer. The other missed blocks in the neighbor peer might be pulled from its neighbor peers.

The push approach is illustrated in Figure 2.21. For each slice in the Window Duration, the neighbor peer periodically tests its neighbors to check for lower latency. The neighbor peer picks the quickest source peer for each slice and sends a push mask to the chosen source peer. In this example, the neighbor peer chooses the peer as the source for slice number 2nd, 5th and 8th. The neighbor peer sends the push mask to the peer. The push mask on the peer is also updated periodically with lower rate than data delivery. If video blocks for the push mask arrive, the corresponding blocks are pushed to the neighbor peer immediately.

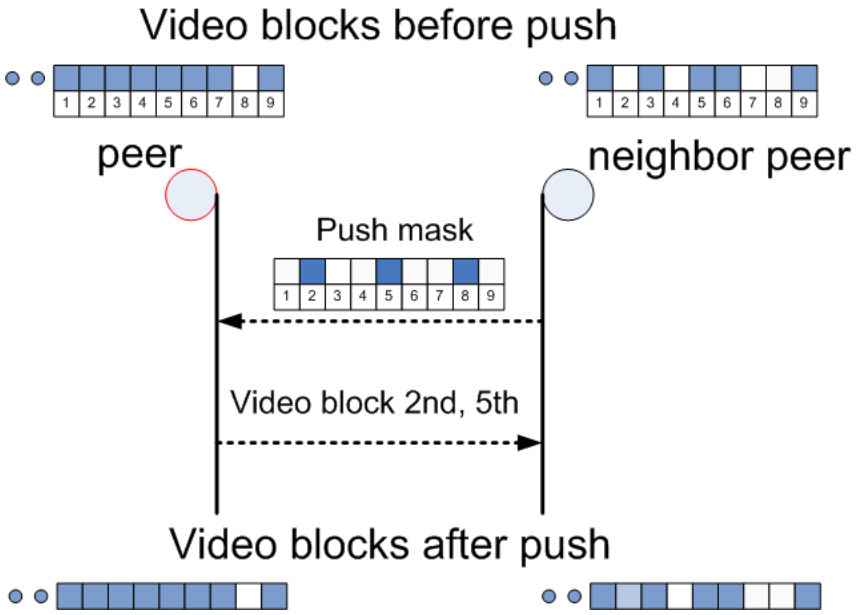


Figure 2.21: The push approach for P2P on the Flash platform.

Adobe introduced the P2P multicast since Flash Player 10.1 and Flash Media Server 4.0. In addition, Adobe provided corresponding ActionScript APIs that allow the developer to create P2P multicast for live streaming services on the Flash platform.

2.3 Summary

In this chapter, we have presented multicast communication at the network layer and at the application layer in detail. Several media streaming systems and their underlying streaming technologies are described. Among them, we find out that Adobe's media streaming system provides possibility to support both IP multicast and P2P multicast based on its new streaming protocol - Real-Time Media Flow Protocol (RTMFP). Therefore, in next chapter we describe the design and implementation of a prototype demonstrator on Adobe's Flash platform.

Chapter 3

Methodology

In last chapter, we figure out the demonstrator is implemented on Adobe's Flash platform. This chapter presents the technologies and tools on Adobe's Flash platform. These technologies and tools that are used for our implementation are described in more detail. Finally, the detailed implementations of the demonstrator's server-side and client-side applications are given.

3.1 Adobe Flash Platform

Adobe Flash Platform is the leading web design and development platform for creating expressive applications, content, and video that run consistently across operating systems and devices and reach over 98% of Internet-connected desktop users. It consists of an integrated set of technologies including client runtimes, tools, frameworks, services, and servers as illustrated in Figure 3.1.

In the implementation of the demonstrator, below Adobe's technologies are included:

- Flash Media Development Server 4.5 - a limited, free version for development of Flash Media Server family.
- Flash Player 10.1 and later - the client runtime for the web browser.
- Flash Builder 4.6 - the tool to develop Flash applications running on Flash Player.

This section first presents the above technologies used in the demonstrator. Some important ActionScript APIs for developing the client-side and server-side applications of the demonstrator are introduced.

3.1.1 Flash Media Development Server 4.5

Flash Media Development Server 4.5 contains all the features in Adobe Flash Media Enterprise Server 4.5 software with some limitations. Table 3.1 lists the limitations related to multicast streaming. It can be used to evaluate and

Figure 3.1: Overview of the Adobe Flash Platform.(from Adobe)

deploy small scale solutions. Its installation file can be downloaded free of charge from Adobe's website[1].

Table 3.1: Feature limitations of Flash Media Development Server 4.5

Features	**Limitations**
IP multicast	10 minutes
RTMFP unicast	50 connections
RTMFP P2P introductions	50 connections
Multicast fusion for Flash Player compatible devices	10 minutes
Application-level multicast	10 minutes
Peer-assisted networking	50

Installation of FMS

We have installed Flash Media Development Server 4.5 on a computer located at Media Network Service AS. The operating system is Linux CentOS 5.5 64 bit with kernel 1.1. Some hardware information of this computer: 2.40 GHz Intel Xeon processor, 4G RAM and 1-GB Ethernet card. This computer is located on the network with domain name *MNSBONE.NET*. This network supports native IP multicast.

[1] http://www.adobe.com

Our installation uses the default options except the installation of the Apache HTTP server. We use the existing Apache HTTP Server Project[2] on this computer and proxy HTTP connections from FMS to this server. With Apache http, the server can serve media, client SWF files, HTML files, and other page-related files over HTTP. After installation, FMS is started automatically. A Flash Media Server Start Screen is launched on the link[3]. This page can be used to check whether FMS is installed and started successfully. Flash Media Administration Server is also installed and connected to FMS. This server listens on port 1111 by default. It also supports the Administrator APIs that can be used to create tools for monitoring and administering FMS.

Administration Console

The Administration Console is an administrative application that calls Administration APIs to inspect and manage the server. It is wrapped into an HTML page and can be launched on the link[4]. After the administrator logs in, he/she can manage FMS and view information about applications running on FMS. Figure 3.2 shows one snapshot of the console after login. In this snapshot, the console displays the live log of a running application *vod*. It can also present the clients, shared objects, streams and performance of this application if other tab is selected.

Configuration of FMS

FMS can be configured by the configuration files located in the *rootinstall/conf* folder. The folder structure is shown in Figure 3.3.

This folder contains two directories and seven files that are described in Table 3.2. These configuration files are in the form of XML. FMS needs to be restarted if any configuration file has been changed.

FMS has a hierarchical structure with four levels: *server*, *adaptor*, *virtual host* and *application*. The *server* is at the top level and contains one or more adaptors. The *adaptor* is assigned an IP address or a port number. The *adaptor* has its own directory inside the *conf* directory. The adaptor directory shall contain an *Adaptor.xml* and a *_defaultVHost_* directory. Each adaptor shall have a default virtual host and other custom virtual hosts. The *virtual host* can not be assigned an IP address or a port number, but must be mapped to a DNS entry or other name resolutions. The virtual hosts are used to host multiple websites on a server, for example, *www.example.com* and *www.test.com* at the same server. FMS uses adaptors to organize virtual hosts by IP address or port number. The virtual host has its own directory inside the adaptor directory. The virtual host directory contains *Vhost.xml* and *Application.xml*. A virtual

[2]http://httpd.apache.org
[3]http://fms.medianetworkservices.com
[4]http://fms.medianetworkservices.com/fms_adminConsole.html

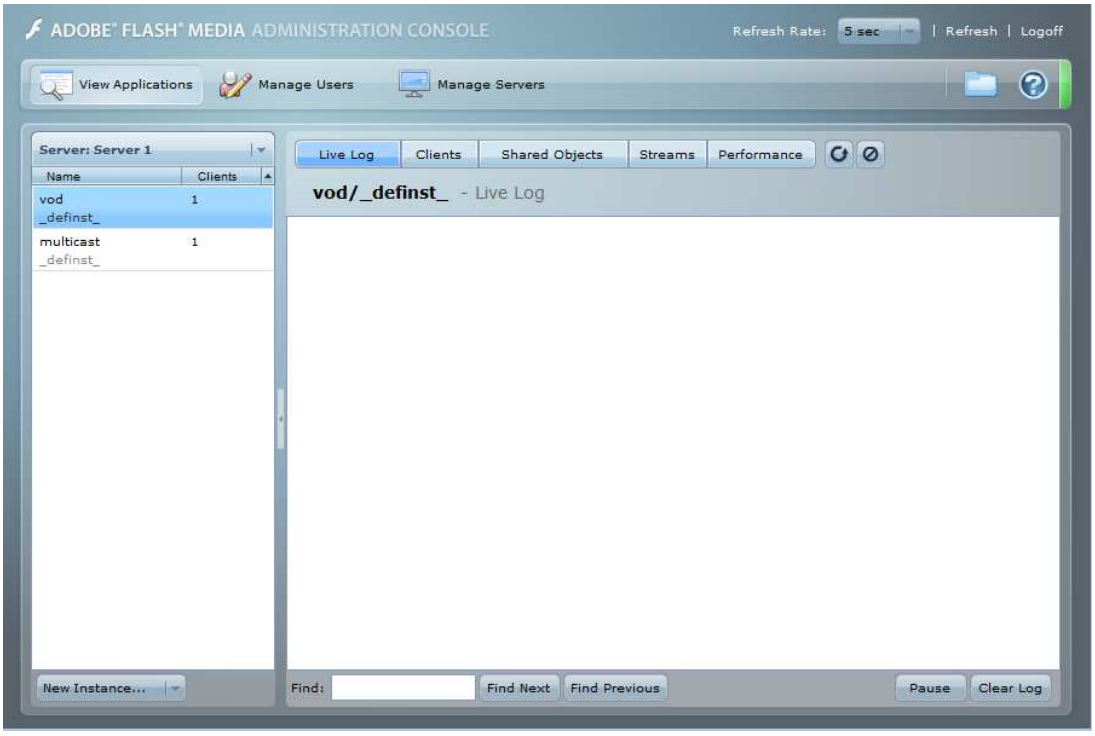


Figure 3.2: Adobe Flash Media Administration Console.

host can host multiple applications. The application directory for the virtual host can be defined in *fms.ini* and in its Vhost.xml. For example, *fms.ini* defines this directory with *VHOST.APPDIR = rootinsall/applications* and *Vhost.xml* in the virtual host directory with *<AppDIR>${VHOST.APPDIR}</AppDIR>*. This directory is registered directory for the server-side applications.

Server-side application

The server-side applications are located in the application directory for the virtual host, for example, the *rootinstall/applications* in this thesis. One server-side application is defined as a directory under this registration directory. The name of the directory is the name of server-side application. The directory usually contains a server-side ActionScript file and other files for configuration. Figure 3.1.1 shows the structures of *live* and *multicast* applications. The two applications are sample server-side applications from Adobe. The server-side ActionScript file is usually *main.asc* or *main.far*. The *main.far* file is the packaged file using *far.exe* tool. In addition, an *Application.xml* can be located in a particular application directory as examples in Figure 3.1.1. This applications-specific file can override the settings in *Application.xml* in the virtual host configuration directory. The settings of a particular application can be changed without restarting the server. The application directory can also include other files if needed. These files can be *.txt*, *.xml* or other formats. These files can be open and read in the ActionScript file.

```
[fms@adobe-1 fms]$ tree conf/
conf/
|-- Logger.xml
|-- Server.xml
|-- Users.xml
|-- _defaultRoot_
|   |-- Adaptor.xml
|   `-- _defaultVHost_
|       |-- Application.xml
|       `-- Vhost.xml
`-- fms.ini

2 directories, 7 files
```

Figure 3.3: Default structure of the configuration folder for FMS.

```
[fms@adobe-1 fms]$ tree applications/multicast/
applications/multicast/
|-- Application.xml
`-- main.asc

0 directories, 2 files
```

(a) multicast application

```
[fms@adobe-1 fms]$ tree applications/live
applications/live
|-- Application.xml
|-- allowedHTMLdomains.txt
|-- allowedSWFdomains.txt
|-- main.far
`-- readme.txt

0 directories, 5 files
```

(b) live application

Figure 3.4: Server-side application directory.

The ActionScript file is the heart of server-side application. When a server-side application is loaded on FMS, the ActionScript file is first loaded. The script is written using Server-Side ActionScript based on ECMAScript edition 3 language specification. FMS contains an embedded JavaScript engine - Mozilla SpiderMonkey that can compile and execute this script. When a client connects to a server-side application on FMS, the script of the application is compiled and a new application instance is created. The function *trace(expression)* is used to debug the script. Its expression appears in the Live Log panel of the Administration Console as shown in Figure 3.2.

3.1.2 Flash Player

The Flash Player can play Flash applications in the form of SWF files and is deployed as plug-in in web browser. The SWF files are completed, complied and published files. These files can not be edited any more. The SWF files in this thesis are built and published using Flash Builder and are wrapped into an HTML page. An example of script block to wrap a *SWFname.swf* is shown in Listing 3.1. In this way, the client can launch the SWF files in the HTML

Table 3.2: Configuration folder on FMS

File/Directory	Description
Logger.xml	a configuration file that contains the elements and information used to configure log files. The information includes the location of the log files, the events written in the log files, etc.
Server.xml	a configuration file which affects the entire server unless that is overridden in another configuration files.
Users.xml	a configuration file which identifies administrators and set their access permissions.
*_defaultRoot_*	a directory that contains configuration files for the default adaptor and configuration folders for the virtual hosts inside it. The name of directory is the name of the adaptor.
Adaptor.xml	a configuration file for the adaptor. It determines the number of threads that can be used by the adaptor, the communications ports the adaptor binds to, and the IP addresses or domains from which the adaptor can accept connections, etc.
*_defaultVHost_*	a directory that contains configuration files for default virtual host. This virtual host is mapped to *fms.medianetworkservices.com*.
Application.xml	a configuration file that defines the default settings for all applications within the virtual host.
Vhost.xml	a configuration file that defines the settings for the virtual host. These settings include aliases for the virtual host, the location of the virtual host's application directory, limits on the resources the virtual host can use, and other parameters.
fms.ini	a default configuration file which contains the most commonly edited configuration parameters.

pages delivered by Apache HTTP server.

Listing 3.1: An example of script from an HTML file that wraps an SWF file

```
<script type="text/javascript">
  // For version detection, set to min. required Flash Player version,
  //or 0 (or 0.0.0), for no version detection.
  var swfVersionStr = "11.1.0";
  // To use express install, set to playerProductInstall.swf,
  //otherwise the empty string.
  var xiSwfUrlStr = "playerProductInstall.swf";
  var flashvars = {};
  var params = {};
  params.quality = "high";
  params.bgcolor = "#ffffff";
  params.allowscriptaccess = "sameDomain";
  params.allowfullscreen = "true";
  var attributes = {};
  attributes.id = "SWFname";
  attributes.name = "SWFname";
  attributes.align = "middle";
  swfobject.embedSWF(
    "SWFname.swf", "flashContent",
    "100%", "1302",
    swfVersionStr, xiSwfUrlStr,
    flashvars, params, attributes);
  // JavaScript enabled so display the flashContent div
  // in case it is not replaced with a swf object.
  swfobject.createCSS("#flashContent", "display:block;text-align:left;");
</script>
```

Flash Player Settings Manager

Flash Player Settings Manager provides options to control applications run on Flash Player. It provides several panels to manage global settings, such as privacy settings, storage settings, security settings, and automatic notification settings, etc. Setting Manager is available locally or online. The Local Setting Manager is accessed on Windows, Mac, and Linux computers with Flash Player version 10.3 and later. For example, it can be accessed in the Control Panel on Windows and in System Preferences on Mac. For other OS and earlier versions of Flash Player, the Online Settings Manager is accessed from Flash Player Helper page[5].

3.1.3 Flash Builder

Adobe Flash Builder is an integrated development environment (IDE) for building cross-platform, rich Internet applications (RIAs) for desktops and a wide variety of mobile devices. Flash Builder is built on top of Eclipse and provides

[5]http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager.html

all the tools required to develop applications using open-source Flex framework and ActionScript 3.0. It can build many kinds of applications that use Flex framework, MXML, Adobe Flash Player, Adobe AIR, ActionScript 3.0, and LiveCycle Data Services. The client-side applications in this thesis are compiled MXML applications using ActionScript 3.0 and MXML.

ActionScript 3.0[13] is an object-oriented programming language ideally for building rich Internet applications. It is based on ECMAScript which is the international standardized programming language for scripting. MXML is an XML-based user interface markup language introduced by Macromedia. The source codes of an example MXML application *test.mxml* are shown in Listing 3.2. This application contains a button and a label. To click the button can change the text of the label.

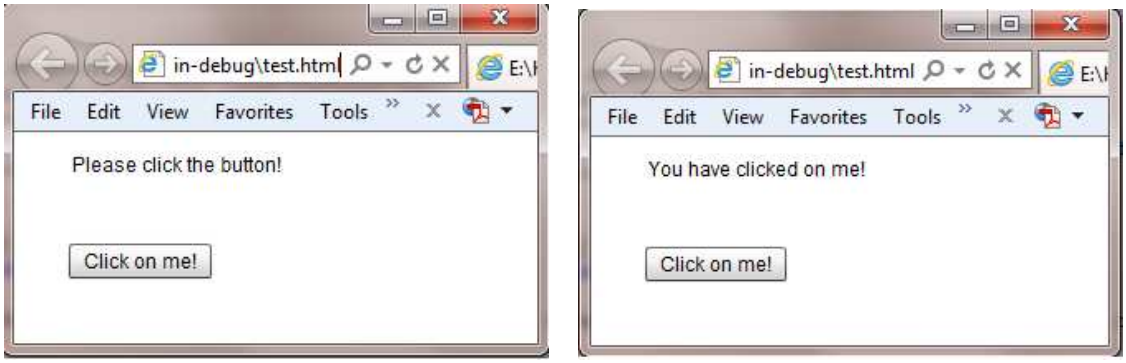
Listing 3.2: The source code of an example MXML application

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
          xmlns:s="library://ns.adobe.com/flex/spark"
          xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955"
          minHeight="600" creationComplete="init()">

  <fx:Script>
    <![CDATA[
      private function init():void
      {
        buttonClick.addEventListener(MouseEvent.CLICK, clickHandler);
      }
      private function clickHandler(e:MouseEvent)
      {
        labelStatus.text = "You have clicked on me!";
      }
    ]]>
  </fx:Script>

  <fx:Declarations>
  <!-- Place non-visual elements (e.g., services, value objects) here -->
  </fx:Declarations>

  <s:Button id="buttonClick" x="33" y="67" label="Click on me!"/>
  <s:Label id="labelStatus" x="33" y="10" width="181" height="49"
    text="Please click the button!"/>
</s:Application>
```

Building this MXML file generates a *test.swf* file and a *test.html* file. The SWF file is wrapped into the HTML file in the same way as shown in Listing 3.1. The HTML file opened in IE browser is shown in Figure 3.1.3.

3.1.4 ActionScript 3.0 APIs

This section presents some classes used in demonstrator's client-side applications referring to ActionScript 3.0 Reference[2].



(a) Before clicking the button (b) After clicking the button

Figure 3.5: The HTML page *test.html* in IE

Camera class

The *Camera* class is used to capture video from the client system or device camera. The captured video can be displayed locally or transmitted. The video quality can be specified by the properties of *Camera*. The following properties are considered in the demonstrator.

- *height*: the capture height in pixels.
- *width*: the capture width in pixels.
- *fps*: the maximum rate at which the camera can capture data, in frames per second.
- *quality*: the required level of picture quality, as determined by the amount of compression being applied to each video frame.
- *bandwidth*: the maximum amount of bandwidth that current outgoing video feed can use, in bytes per second.

Video class

The *Video* class is used to display live or recorded video. The video can be the recorded video on a server or locally, or the live video captured from camera.

NetConnection Class

The *NetConnection* class creates a two-way connection between a client and an application instance on the server, or a two-way network endpoint for RTMFP P2P group communication. It can establish the connection, but can not send streams of media and data. Another class *NetStream* is used to deliver streams over the connection.

NetStream Class

The *NetStream* class is used to open a one-way streaming channel over established connection. A streaming channel is unidirectional and can carry more than one type of content. The content can be audio, video and message data.

NetGroup Class

The *NetGroup* class is used to form and represent an RTMFP P2P group.

GroupSpecifier Class

The *GroupSpecifier* class is used to define the parameters and capabilities of an RTMFP P2P group. The important parameters and capabilities for a multicast-enabled group include:

- group name;
- whether multicast streaming is enabled;
- whether peer-to-peer connections are disabled;
- whether members can open a channel to the server;
- IP multicast address if IP multicast is enabled. This address causes the member to join the specified IP multicast group and listen to the specified UDP port.

Once such a object is created, an opaque *groupspec* string can be created using the method *GroupSpecifier.groupspecWithAuthorizations()*. This string starts with "*G* :" followed by hexadecimal digits, for example "*G* : 01010*b*...". If any parameter or capability of *GroupSpecifier* is changed, the new string is created and represents a new group.

MulticastStreamInfo Class

A *NetStream* object with underlying RTMFP P2P and IP multicast stream transport is named as a multicast stream. The *MulticastStreamInfo* class presents various Quality of Service (QoS) statistics of a multicast stream on a local node with the following properties:

- the number of media bytes sent or received,
- the number of media fragments sent or received,
- the number of control bytes sent or received,
- the snapshot of the current rate averaged over a few seconds.

NetStatusEvent Class

The *NetStatusEvent* class is a subclass of the *Event* class. A *NetStatusEvent* object is dispatched when a *NetConnection*, *NetStream*, or *SharedObject* object reports its status or its error condition. The status or error is represented using a string contained in *NetStatusEvent* object. For example, when attempting to establish a connection by calling *NetConnection.connect()*, a *NetStatusEvent* object is returned. If the string contained is "*NetConnection.Connect.Success*", it means that the connection attempt has succeeded. If the string is "*NetConnection.Connect.Failed*", the connection attempt has failed.

3.1.5 Server-Side ActionScript APIs

This part introduces some classes used in server-side script *multicast/main.asc* referring to Adobe's Server-Side ActionScript Language Reference[1].

Application Class

The *Application* class is used to represent an instance of a Flash Media Server application. It is created automatically when an application instantiated by the server. It can be used to accept and reject client connection attempts, to register and un-register classes and proxies, and to manage the life cycle of an application.

Client Class

The *Client* class is used to handle each user which is connected to a Flash Media Server application instance. It is created automatically when a user connects to an application on the server. It is destroyed when the user disconnects from the application. An application instance is allowed to have thousands of active clients connected. This class allows the server to determine the properties of each client, such as its version, its platform, and its IP address, etc. It also provides the methods to set bandwidth limits and to call methods in client-side scripts.

NetConnection Class

The server-side *NetConnection* class is used to create a two-way connection between a Flash Media Server application instance and an application server, another Flash Media Server, or another Flash Media Server application instance on the same server.

NetStream Class

The server-side *NetStream* class is used to open a one-way channel for publishing a stream to a remote Flash Media Interactive Server, an multicast group

and an RTMFP group. The multicast group is specified using an multicast address and a port.

NetGroup and GroupSpecifier Classes

The two classes for server-side application are the same as these for client-side application. The *GroupSpecifier* class is used to describe an RTMFP group. The *NetGroup* class is used to represent membership of the RTMFP group.

3.2 Environments for Demonstrator

This thesis implements a demonstrator which provides real-time video service on the Flash platform. This demonstrator uses Adobe's new protocol - Real Time Media Flow Protocol (RTMFP) for streaming. The demonstrator also utilizes Adobe's P2P technology - RTMFP P2P Group. Therefore, the environments around demonstrator must support RTMFP and RTMFP P2P group. This section presents some important settings for supporting these two technologies.

3.2.1 Configure Flash Media Server for RTMFP

FMS's configuration files allow the user to specify the parameters and ports for RTMFP connections and RTMFP groups.

Parameters configuration

To enable an RTMFP connection on FMS, the attribute of *RTMFP* must be set to be "*true*" with the following markup in *rootinstall/conf/_defaultRoot_/Adaptor.xml* file: *<RTMFP enable="true"> </RTMFP>*. The contents inside this markup specify the parameters for an RTMFP connection, including the frequency in seconds at which stats are updated for the RTMFP connection, the frequency in seconds at which RTMFP keep alive packets are sent, the method in which RTMFP should deal with forwarded request message and the redirect ports.

Another configuration file *rootinstall/conf/_defaultRoot_/_defaultVHost_/Application.xml* includes the *<RTMFP></RTMFP>* markup. The contents inside this markup specify the means to control the behavior of application-specific RTMFP functionality, such as controlling server-side functionality corresponding to the server channel, controlling whether events for clients that join or leave a Group are dispatched to server-side script or handled internally, configuring how the server handles peer lookup events.

Port configuration

To establish an RTMFP connection between client and Flash Media Server, the following ports must be available on the server running Flash Media Server.

- *UDP port 1935*: Flash Media Server listens to RTMFP request over this port. This port can be specified in the **ADAPTOR.HOSTPORT** parameter in the *rootinstall/conf/fms.ini* file.
- *UDP ports 19350-65535*: These ports are called RTMFP redirect ports or RTMFP migration ports. Flash Media Server binds one port in this range to RTMFP listener and sends the port number to RTFMP client. Afterwards the client connects to Flash Media Server over this port. This port range can be specified in the *rootinstall/conf/_defaultRoot_/Adaptor.xml* file.

3.2.2 RTMFP Connectivity

NAT (network address translation) and firewall filtering can block RTMFP connections. To ensure a client on a particular network can create an RTMFP connection, the network must satisfy the following requirements:

- Allow UDP traffic. RTMFP is based on UDP.
- NAT or firewall used has predictable behavior. It complies with the NAT implementation recommendations of IETF BEHAVE working group[6].

In addition, Matthew Kaufman, the inventor of RTMFP provides a website called *RTMFP Connectivity Checker*[7]. A client opens this website in a web browser. It checks whether this client can establish RTMFP connection to **Adobe Chicago IL USA server**. It also indicates the behaviors of NAT and firewall filtering of the network where the client locates.

This checker does the following tests:

- *Knows public IP address of self.* This checks whether there is address translation.
- *Public UDP port number the same as local UDP port number.* This checks whether there is port translation. If there are neither address translation nor port translation, no NAT exists.
- *Can receive from same IP address, same UDP port number.* This checks whether NAT and firewall filtering is static. This answer should always be *YES* because it is mandatory for RTMFP establishing initial connection.

[6]http://tools.ietf.org/wg/behave/
[7]http://cc.rtmfp.net

- *Can receive from same IP address, different UDP port number.* This checks whether NAT and firewall filtering is port restricted.

- *Can send to different IP address, different UDP port number.* This checks whether NAT and firewall filtering is address restricted.

- *Can send to different IP address after server introduction.* This test is like opening a new RTMFP connection after the initial connection. This answer should be *YES* if the initial connection can be made.

- *Source IP address is preserved from original connection.* This checks the type of NAT. It can be one of the following: a cone NAT, a symmetric NAT with single IP address, a symmetric NAT with multiple IP addresses but the same address happens to be used this time. If running checker repeatedly and the values change, NAT is symmetric and with multiple IP addresses.

- *Source UDP port number is preserved from original connection.* This checks whether an NAT is a cone NAT or a symmetric NAT.

3.2.3 Peer-assisted Networking Setting of Flash Player

The demonstrator allows sharing video among an RTMFP group. Each member has to enable P2P uplink to share its bandwidth with others. So it is necessary to enable peer-assisted networking setting in Flash Player. Flash Player Setting Managers has a peer-assisted networking setting panel. Figure 3.6 shows this panel for Windows 7. Using this panel allows Flash Player to share bandwidth with user's permission for one website.

There are two ways to enable this option. One is to allow the specified website to share bandwidth all the time without asking. For example, the highlighted line in Figure 3.6 has set its Network Access to "*Allow*". Another way is to set Network Access of a website with "*Ask me*". In this case, Flash Player will ask the user before sharing bandwidth using a pop-out window.

3.3 Demonstrator

This thesis is meant to implement a demonstrator that delivers live video to multiple clients using multicast. Our prototype demonstrator is built on Flash platform and is accessed using the link [8]. It consists of publisher, receiver, *multicast* application on Flash Media Server and Apache HTTP Server. Figure 3.7 shows an example of the working demonstrator with one receiver. For a video streaming, the demonstrator can only have one publisher, but multiple receivers are allowed.

[8] http://fms.medianetworkservices.com/multicast_demonstrator/

Figure 3.6: Peer-assisted networking setting panel of Flash Player.

The publisher and the receiver are two computers opening HTML pages *RTMP_Publusher.html* and *RTMFP_Receiver.html* respectively. These two pages wrap *RTMP_Publusher.swf* and *RTMFP_Receiver.swf* respectively. The SWF files are compiled Flash applications that are programmed using MXML and ActionScript 3.0 in Flash Builder. Their detailed implementations are presented in Section 3.3.2 and Section 3.3.3. The publisher connects to *multicast* application on FMS over an RTMP connection. It captures live video from the connected web camera and sends this video to *multicast* application on FMS. The receiver connects to *multicast* application on FMS over an RTMFP connection. It joins this RTFMP group, receives video published to this group and then plays it. The RTMFP group is specified using RTMFP group specifier described in Section 3.3.1.

The *multicast* application on FMS is a server-side application provided by Adobe. Its tasks are to receive video from the publisher and publish this video to an RTMFP group. A description of this application is presented in Section 3.3.4.

3.3.1 RTMFP Group Specifier

An RTMFP group can be defined using the *GroupSpecifier* class. The demonstrator allows the user to define the group with one of the following multicast types.

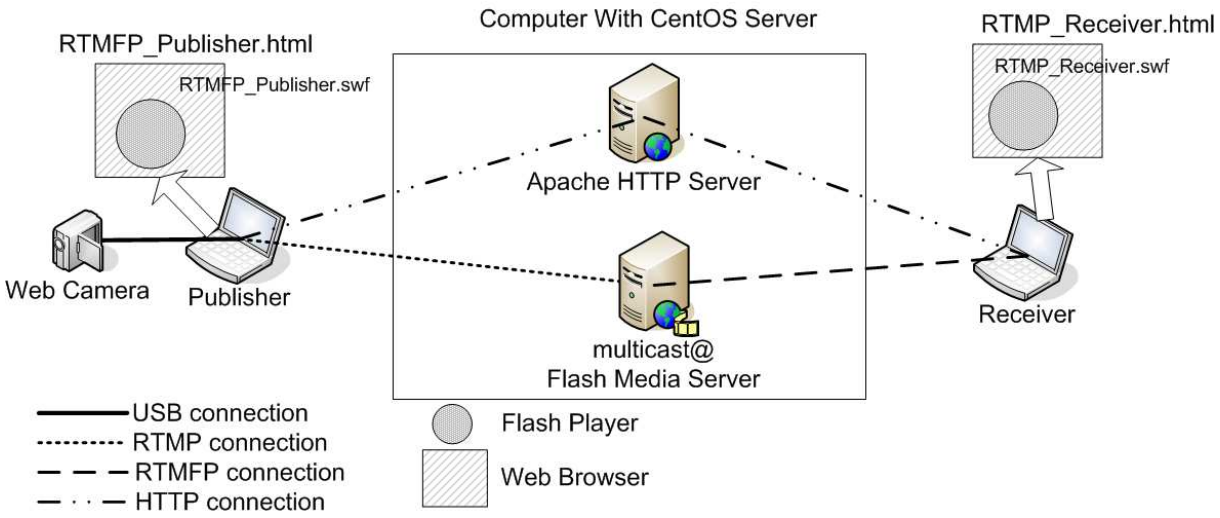


Figure 3.7: An example of the working demonstrator: it consists of a web camera, a publisher, a receiver, *multicast* applications on FMS and Apache HTTP Server. The computer opening *RTMP_Publusher.html* is named as a publisher. The publisher captures live video from web camera and sends this video over an RTMP connection. The computer opening *RTMFP_Receiver.html* is names as a receiver. The receiver obtains video over RTMFP connection, and then plays it. The *multicast* application is a server-side application on FMS. It receives video from the publisher and publishes this video to an RTMFP group. The Apache HTTP Server delivers *.html* and *.swf* files over HTTP connections.

P2P multicast

The ActionScript codes to define parameters for a group using only P2P multicast are in Listing 3.3.

Listing 3.3: Parameters of an RTMFP group using only P2P multicast

```
GroupSpecifier groupspec = new GroupSpecifier(groupName);
groupspec.serverChannelEnabled = true;
groupspec.multicastEnabled = true;
groupspec.setPublishPassword(publishPwd);
groupspec.peerToPeerDisabled = false;
```

IP multicast

The ActionScript codes to define parameters for a group using only IP multicast are in Listing 3.4.

Listing 3.4: Parameters of an RTMFP group using only IP multicast

```
GroupSpecifier groupspec = new GroupSpecifier(groupName);
groupspec.serverChannelEnabled = true;
groupspec.multicastEnabled = true;
groupspec.setPublishPassword(publishPwd);
groupspec.peerToPeerDisabled = true;
groupspec.addIPMulticastAddress(multicastAddr);
groupspec.ipMulticastMemberUpdatesEnabled = true;
```

Fusion multicast

A group with fusion multicast indicates that video delivery can use IP multicast and P2P multicast coordinated and concurrently. IP multicast will be preferred. The ActionScript codes to define parameters for the group are in Listing 3.5.

Listing 3.5: Parameters of an RTMFP group using fusion multicast

```
GroupSpecifier groupspec = new GroupSpecifier(groupName);
groupspec.serverChannelEnabled = true;
groupspec.multicastEnabled = true;
groupspec.setPublishPassword(publishPwd);
groupspec.peerToPeerDisabled = false;
groupspec.addIPMulticastAddress(multicastAddr);
groupspec.ipMulticastMemberUpdatesEnabled = true;
```

3.3.2 Publisher

The publisher opens the HTML page *RTMP_Publisher.html* in a web browser. Its major tasks are to capture live video from camera and send this video to FMS over an RTMP connection.

GUI

When opening the publisher on a web browser, its starting GUI is loaded as shown in Figure 3.8. It has a panel *Settings for Publisher*. On this panel, the user can specify the name or IP address of Flash Media Server and its server-side application that the publisher is connecting to. There are two tools to configure the video published. One tool is *Multicast Config Tool*. It is used to specify the parameters of an RTMFP group. These parameters include multicast type, stream name, group name, publish password, and IP multicast address if IP multicast is enabled. Another tool is *Video Quality Tool*. It is used to choosing published video's resolution. This demonstrator allows the user to choose a video resolution among 160×120, 320×240, 640×480 and 800×600 pixels.

The publisher starts video streaming by clicking the *PUBLISH* button. The streaming GUI is loaded as shown in Figure 3.9. There is no *Settings for Publisher* panel for configuration. The video captured from camera is displayed. The user can click the *STOP* button to stop video streaming and come back to the starting GUI.

In both GUIs, there is a *Status Window* that shows the texts specified in AcitonScript codes of publisher. This text window is used as debugging tool in our programming. These texts contain information about invoked NetStatusEvent events. A successful publishing as shown in 3.9 has invoked the following events in order:

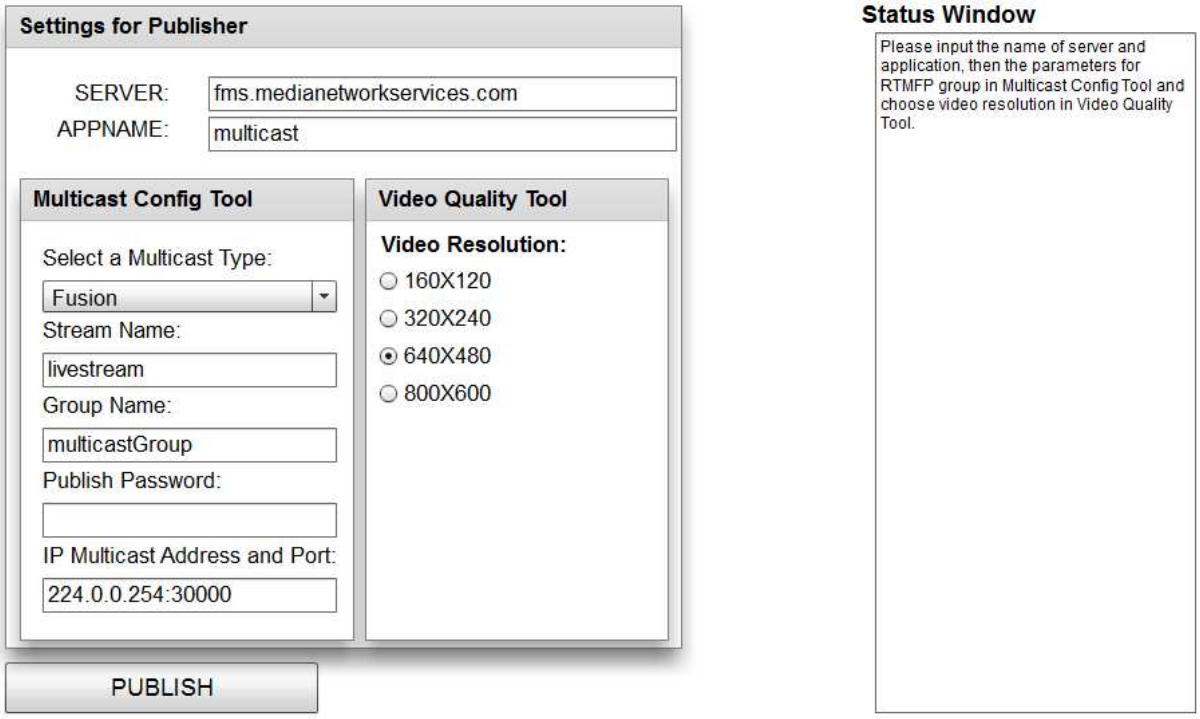


Figure 3.8: Starting GUI of publisher with *Settings for Publisher* panel.

1. *NetConnection.Connect.Success*,

2. *NetStream.Publish.Start*.

The last event implies that video starts to be sent.

Camera setting

The *Camera* class allows the developer to set the camera capture mode. In this demonstrator, only video resolution can be changed on publisher's starting GUI. For other camera settings, the publisher uses the following default values:

- $fps = 15$ frames per second.

- $quality = 100$. This value means the highest picture quality and no compression being applied to each video frame.

- $bandwidth = 0$. This value indicates that quality takes precedence. The runtime uses as much bandwidth as required to maintain the specified quality. If necessary, the runtime reduces the frame rate to maintain picture quality. In general, as motion increases, bandwidth usage also increases.

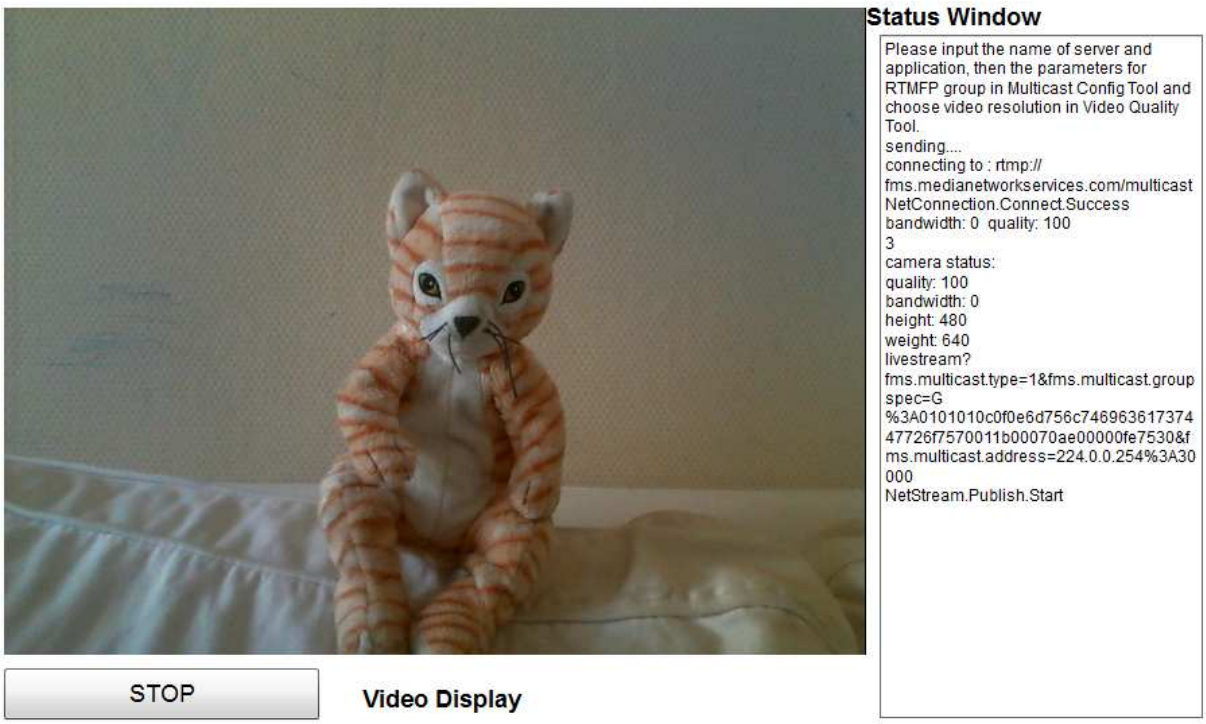


Figure 3.9: Streaming GUI of the publisher with video captured from camera.

ActionScript codes

The actions behind the GUI are programmed using ActionScript 3.0 language. The scripts in this section are not the complete program used for the publisher. They are code fragments explaining the core operations of the publisher. By clicking the *PUBLISH* button, the publisher first establishes an RTMP connection with *multicast* application at FMS using the scripts in Listing 3.6.

Listing 3.6: The publisher establishes an RTMP connection with a server-side application at FMS.

```
NetConnection netConnection = new NetConnection();
// Create an event listener to NetStatusEvent related to NetConnection.
netConnection.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
netConnection.connect("rtmp://" + SERVER + "/" + APPNAME);
```

When *NetConnection.connect()* is called, the status of connection is reported as a *NetStatusEvent* object. If the information in the object contains *NetConnection.Connect.Success*, the publisher can start sending the video captured from camera using the scripts in Listing 3.7.

Listing 3.7: The publisher starts sending video.

```
private function netStatusHandler(event:NetStatusEvent):void
{
 sitch(event.info.code) {
  case "NetConncetion.Connect.Success":
   NetStream netStream = new NetStream(netConnection);
   // Attach camera to stream for sending.
   netStream.addCamera(camera);
   netStream.publish(publishName);
   break;
 }
}
```

This sending video will be re-published to an RTMFP group by *multicast* application on FMS. The group is specified by the publisher. The information of group is sent in the name of sent stream *publishName* in Listing 3.7. The format of *publishName* is listed in Listing 3.8.

Listing 3.8: The name of sent stream from the publisher.

```
var publishName:String = streamName +
      "?fms.multicast.type=multicastType" +
      "&fms.multicast.groupspec=" +
      escape(groupspec.groupspecWithAuthorizations()) +
      "&fms.multicast.address=" + // only if IP multicast is enabled.
      escape(multicastAddr);
```

The string *streamName* is the name of stream published to RTMFP group. The string *multicastType* has value among 1, 2 and 3 that represent respectively Fusion, IP and P2P multicast. The string generated by *escape(groupspec.groupspecWithAuthorizations())* describes the parameters of RTMFP group. The *groupspec* is a GroupSpecifier object defined as Section 3.3.1. If IP multicast is enabled, the string from *escape(multicastAddress)* contains IP multicast group address and port number.

3.3.3 Receiver

The receiver opens the HTML page *RTMFP_Receiver.html* in a web browser. Its major tasks are to receive the video published for an RTMFP group and play it.

GUI

When opening the receiver on a web browser, its starting GUI is loaded as shown in Figure 3.10. It has a *Settings for Receiver* panel. On this panel, the user can specify the name or IP address of Flash Media Server and its server-side application that the receiver is connecting to. There is a *Multicast Config Tool*. Similar to the tool in publisher, it is used to specify the parameters of an

RTMFP group. To obtain video published, the group specification shall be the same as that in the publisher.

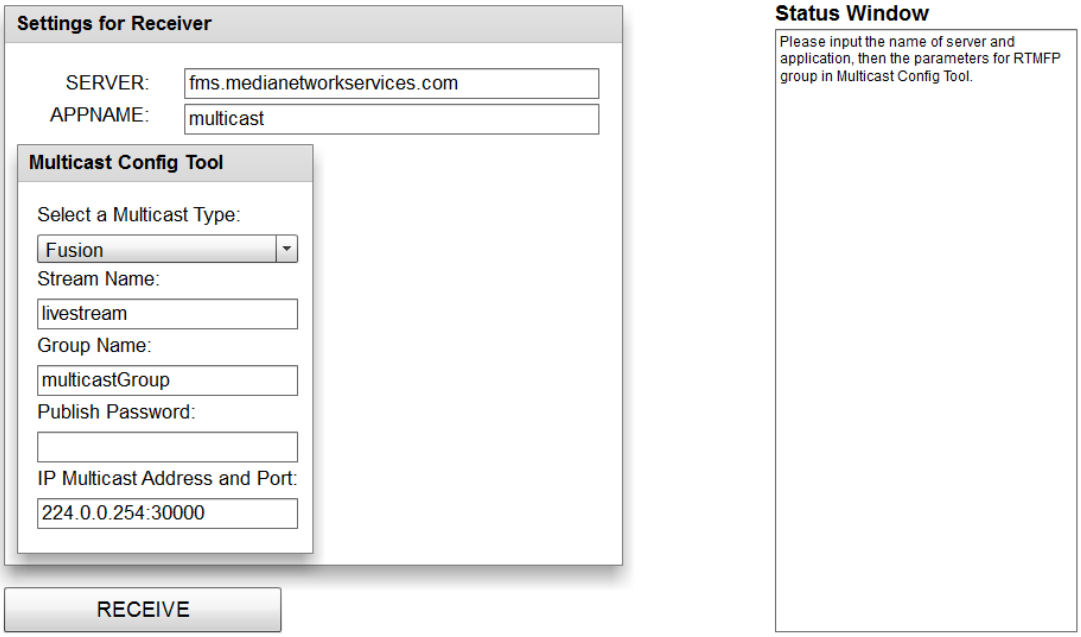


Figure 3.10: Starting GUI of receiver with *Settings for Receiver* panel.

The receiver starts video streaming by clicking the *RECEIVE* button. The streaming GUI is loaded as shown in Figure 3.11. There is no *Settings for Receiver* panel. The obtained video is displayed. The receiving and sending transmission rates are illustrated by the line plots. The rates include media data and control overhead rates at this receiver. The user can click the *STOP* button to stop video streaming and come back to the starting GUI.

As the publisher, both GUIs contain a *Status Window*. A successful receiving as shown in Figure 3.11 has invoked the following NetStatusEvent events in order:

1. *NetConnection.Connect.Success*,
2. *NetGroup.Connect.Success*,
3. *NetStream.Play.Reset*,
4. *NetStream.Play.Start*,
5. *NetStream.Connect.Success*,
6. *NetGroup.Neighbor.Connect*,
7. *NetGroup.MulticastStream.PublishNotify*,
8. *NetStream.MulticastStream.Reset*,

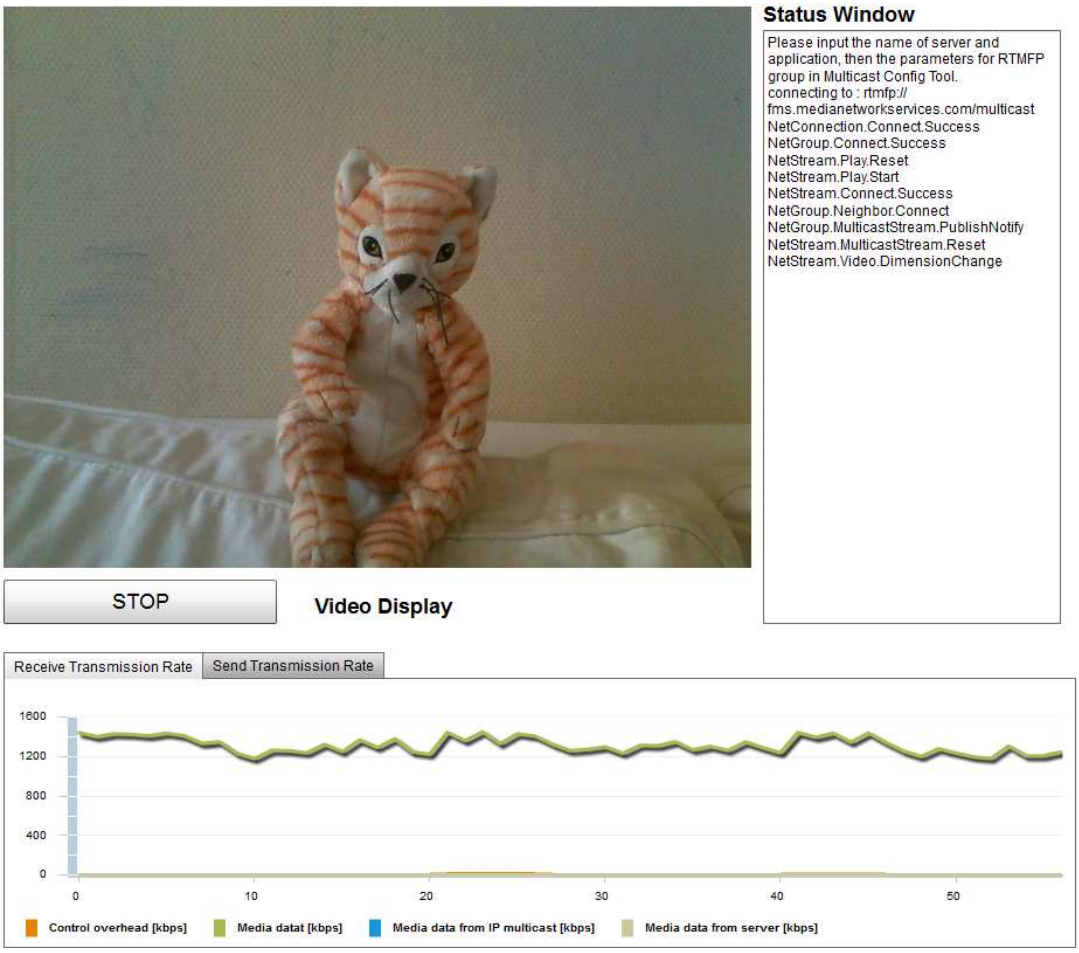


Figure 3.11: Streaming GUI of receiver with *Transmission Rate* charts.

9. *NetStream.Video.DimensionChange*.

The receiver in this demonstrator obtains video only by multicast communication, i.e. as a multicast stream. The video starts to play after the multicast stream has been received.

ActionScript codes

As for the publisher, this section introduces some code fragments explaining the core operations of the receiver. By clicking *RECEIVE* button, the receiver first establishes an RTMFP connection with *multicast* application at FMS using the scripts in Listing 3.9.

Listing 3.9: The receiver establishes an RTMFP connection with a server-side application at FMS

```
NetConnection netConnection = new NetConnection();
// Create an event listener to NetStatusEvent related to NetConnection.
netConnection.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
netConnection.connect("rtmfp://" + SEVER + "/" + APPNAME);
```

When *NetConnection.connect()* is called, the status of connection is reported as a *NetStatusEvent* object. If the information in the object contains *NetConnection.Connect.Success*, the receiver joins the RTMFP group using the scripts in Listing 3.10. The group is described with a string *groupspec*. This string is created using *GroupSpecifier.groupspecWithAuthorizations()*.

Listing 3.10: The receiver joins the RTMFP group.

```
private function netStatusHandler(event:NetStatusEvent):void
{
  switch(event.info.code)
  {
  case "NetConncetion.Connect.Success":
    NetGroup netGroup = new NetGroup(netConnection, groupspec);
    // Create an event listener to NetStatusEvent related to the status
    // of NetGroup.
    netGroup.addEventListener(NetStatusEvent.NET_STATUS,
    netStatusHandler);
    break;
  }
}
```

When a *NetGroup* is created on the *NetConnection* and associated with the *groupspec* string, the status of *NetGroup* is reported as a *NetStatusEvent* object. If the information in the object contains *NetGroup.Connect.Success*, the receiver starts to receive and play the stream published to the group using the scripts in Listing 3.11. *NetStream* is created on *NetConnection* and associated with the *groupspec* string. It indicates this stream is related to the specified RTMFP group.

Listing 3.11: The receiver receives and plays the stream published to the RTMFP group.

```
private function netStatusHandler(event:NetStatusEvent):void
{
 switch(event.info.code) {
  case "NetGroup.Connect.Success":
   NetStream netStream = new NetStream(netConnection, groupspec);
   netStream.play(streamName);
   break;
  }
}
```

3.3.4 Server-side application

The server-side application in the demonstrator is the sample application *multicast* located in the *rootinstall/applications/multicast* folder. This application is available with default installation of FMS. This folder consists of a configuration file *Application.xml* and an ActionScript file *main.asc* as shown in Figure 3.4(a).

Application.xml configures some settings for this application and overrides the corresponding settings in *Application.xml* in the virtual host level. One of

important settings is to turn off live queuing and aggregate messages. This can reduce delay in video's playback.

main.asc is an ActionScript file written in Adobe's Server-Side ActionScript language. It accepts published stream over an RTMP conneciton, and then re-publishes this stream to an RTMFP group and/or an IP multicast group. The RTMFP group specification and the IP multicast group address are obtained from the stream name of published video as listed in Listing 3.8. When Publisher connects to this application and publishes a stream, *main.asc* handles the received stream as illustrated in Figure 3.12.

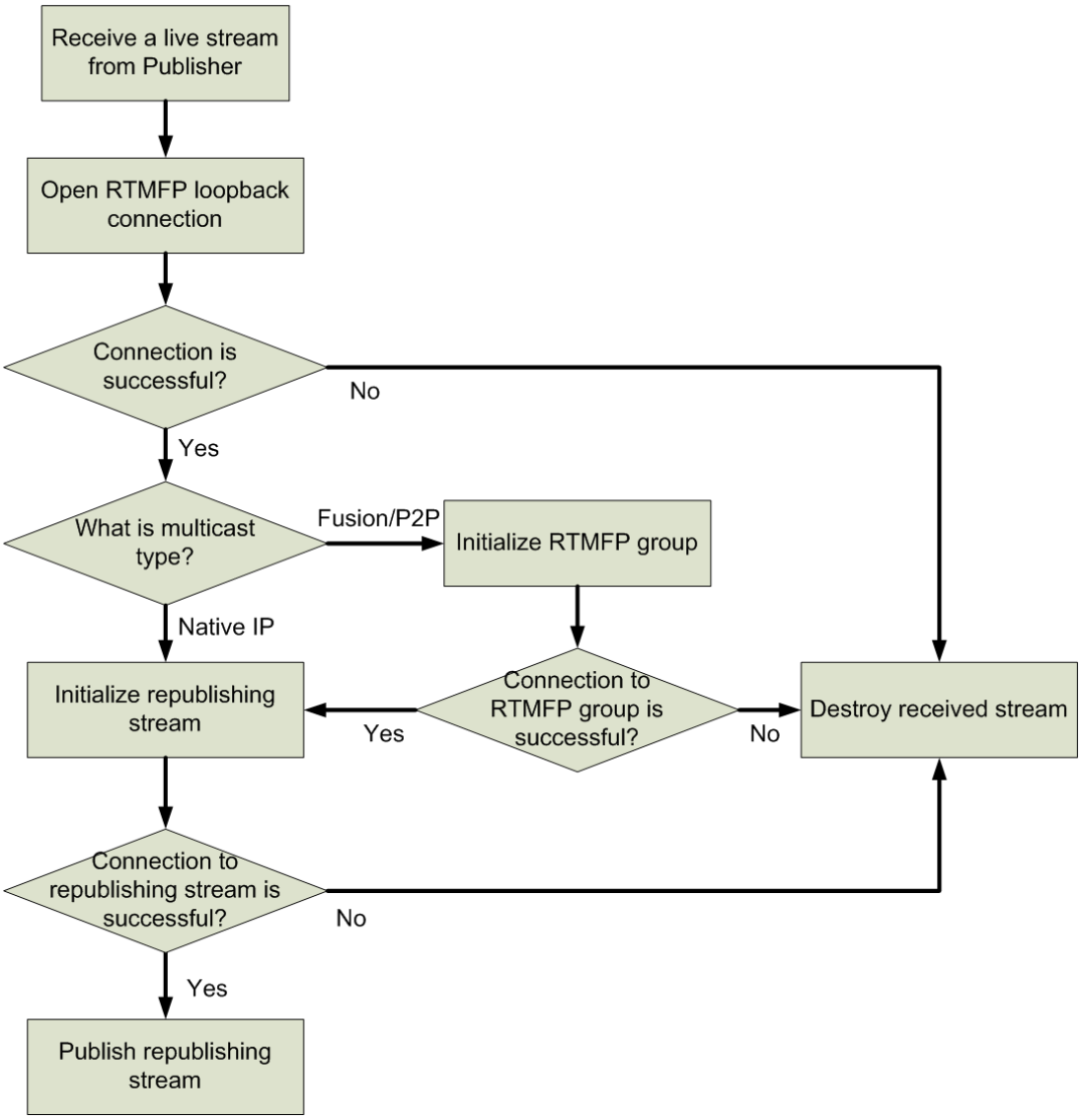


Figure 3.12: Diagram of how server-side script republishes stream to an RTMFP group.

3.4 Summary

In this chapter, we have presented our implementation of a prototype demonstrator on Adobe's Flash platform. It delivers captured live video to multiple clients using multicast communications. At present, this demonstrator is available at the link[9]. In next chapter, a series of experiments using this demonstrator in Internet are performed. Internet traffic are measured and analyzed.

[9] http://fms.medianetworkservices.com/multicast_demonstrator/

Chapter 4

Experiments and Results

A series of experiments of running the demonstrator in real world Internet are performed. The Internet traffic is measured and analyzed.

4.1 Case One: One Publisher and One Receiver

In this measurement scenario, a publisher and a receiver are running respectively on two computers located in home network. This measurement scenario is illustrated in Figure 4.1. The RTMFP group uses P2P multicast. Four experiments with different video resolution are performed. The packets on the publisher and the receiver are captured using Wireshark.

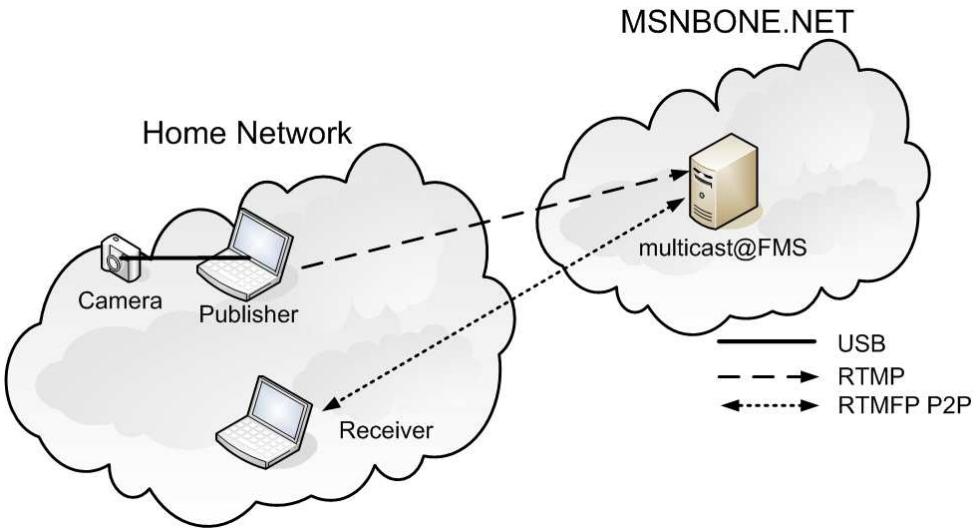


Figure 4.1: Measurement scenario for case one: one receiver and one publisher. A publisher and a receiver are running respectively on two computers located in home network. FMS is running on computer located in the multicast-enabled network.

4.1.1 Publisher

The publisher first establishes an RTMP connection with FMS. Then it starts to deliver video data. This procedure is shown in Figure 4.2.

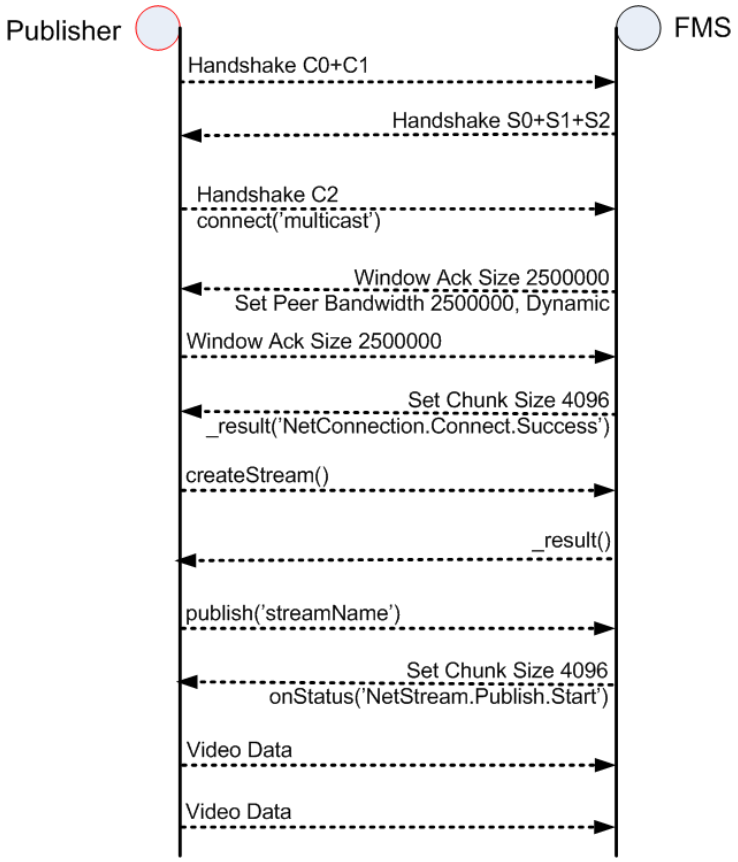


Figure 4.2: The procedure of RTMP connection establishment and streaming from the publisher. First an RTMP connection between the publisher and FMS is established using Handshake packets. Then the publisher starts to deliver video data.

After the connection is established, the packets exchanged over the RTMP link between the publisher and FMS are observed. All packets from FMS to the publisher are Acknowledgements to video data segments. The size of these packets is fixed to be 54 bytes.

A segment of the packets from the publisher to FMS is shown in Figure 4.3. It is observed that there are both RTMP packets and TCP packets.

The RTMP packets are encapsulated in the TCP packets. The type of these RTMP packets is Video Data. An RTMP video data packet consists of header and body. The structure is shown in Figure 4.4. The size of header is 8 bytes. The body starts with one control byte and the rest is video data. The control byte indicates the type and format of the video. In our measurement, the information extracted from the control byte shows that the video encoding format is Sorenson H.263, also known as Sorenson Spark. Sorensen H.263 is an incomplete implementation of H.263. Sorensen H.263 is one of the three video compression formats for Flash video. The other two are H.264 and VP6. There

No.	Time	Source	Destination	Protocol	Length	Info
2267	21.207370	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=294215 Ack=3582 Win=16856 Len=1460
2268	21.207384	192.168.0.101	93.93.96.91	RTMP	261	Video Data
2269	21.239250	192.168.0.101	93.93.96.91	RTMP	1471	Video Data
2272	21.309913	192.168.0.101	93.93.96.91	RTMP	1430	Video Data
2274	21.407082	192.168.0.101	93.93.96.91	RTMP	1459	Video Data
2275	21.438921	192.168.0.101	93.93.96.91	RTMP	995	Video Data
2278	21.511364	192.168.0.101	93.93.96.91	RTMP	1317	Video Data
2280	21.605961	192.168.0.101	93.93.96.91	RTMP	1210	Video Data
2281	21.639043	192.168.0.101	93.93.96.91	RTMP	1233	Video Data
2323	21.711034	192.168.0.101	93.93.96.91	RTMP	1033	Video Data
2324	21.805978	192.168.0.101	93.93.96.91	RTMP	1240	Video Data
2328	21.910656	192.168.0.101	93.93.96.91	RTMP	592	Video Data
2329	21.942842	192.168.0.101	93.93.96.91	RTMP	1134	Video Data
2332	22.009316	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=308402 Ack=3582 Win=16856 Len=1460
2333	22.009327	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=309862 Ack=3582 Win=16856 Len=1460
2334	22.009335	192.168.0.101	93.93.96.91	TCP	1230	57350 > macromedia-fcs [PSH, ACK] Seq=311322 Ack=3582 Win=16856 Len=1176
2335	22.009680	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=312498 Ack=3582 Win=16856 Len=1460
2336	22.009690	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=313958 Ack=3582 Win=16856 Len=1460
2337	22.009695	192.168.0.101	93.93.96.91	TCP	1230	57350 > macromedia-fcs [PSH, ACK] Seq=315418 Ack=3582 Win=16856 Len=1176
2338	22.009909	192.168.0.101	93.93.96.91	RTMP	528	Video Data
2365	22.111364	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=317068 Ack=3582 Win=16856 Len=1460
2366	22.111375	192.168.0.101	93.93.96.91	RTMP	124	Video Data
2386	22.143127	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=318598 Ack=3582 Win=16856 Len=1460
2387	22.143138	192.168.0.101	93.93.96.91	RTMP	202	Video Data
2391	22.207067	192.168.0.101	93.93.96.91	RTMP	1328	Video Data
2395	22.311047	192.168.0.101	93.93.96.91	RTMP	1459	Video Data
2396	22.342928	192.168.0.101	93.93.96.91	RTMP	1046	Video Data
2398	22.407262	192.168.0.101	93.93.96.91	RTMP	1358	Video Data
2401	22.509800	192.168.0.101	93.93.96.91	RTMP	1286	Video Data
2402	22.543121	192.168.0.101	93.93.96.91	RTMP	1418	Video Data
2404	22.606897	192.168.0.101	93.93.96.91	RTMP	1210	Video Data
2447	22.711111	192.168.0.101	93.93.96.91	RTMP	1369	Video Data
2448	22.742913	192.168.0.101	93.93.96.91	RTMP	768	Video Data
2450	22.806973	192.168.0.101	93.93.96.91	RTMP	1151	Video Data
2454	22.910110	192.168.0.101	93.93.96.91	RTMP	1208	Video Data
2456	23.006983	192.168.0.101	93.93.96.91	RTMP	1177	Video Data
2476	23.041490	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=334336 Ack=3582 Win=16856 Len=1460
2477	23.041516	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=335796 Ack=3582 Win=16856 Len=1460
2478	23.041521	192.168.0.101	93.93.96.91	TCP	1230	57350 > macromedia-fcs [PSH, ACK] Seq=337256 Ack=3582 Win=16856 Len=1176
2479	23.041869	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=338432 Ack=3582 Win=16856 Len=1460
2480	23.041878	192.168.0.101	93.93.96.91	TCP	1514	57350 > macromedia-fcs [ACK] Seq=339892 Ack=3582 Win=16856 Len=1460
2481	23.041884	192.168.0.101	93.93.96.91	TCP	1230	57350 > macromedia-fcs [PSH, ACK] Seq=341352 Ack=3582 Win=16856 Len=1176
2482	23.042102	192.168.0.101	93.93.96.91	RTMP	678	Video Data
2504	23.111322	192.168.0.101	93.93.96.91	RTMP	1207	Video Data
2513	23.206720	192.168.0.101	93.93.96.91	RTMP	1420	Video Data
2514	23.238814	192.168.0.101	93.93.96.91	RTMP	988	Video Data

Figure 4.3: A segment of the packets delivered from the publisher to FMS.

are three different types of video data: keyframe, inter-frame and disposable inter-frame.

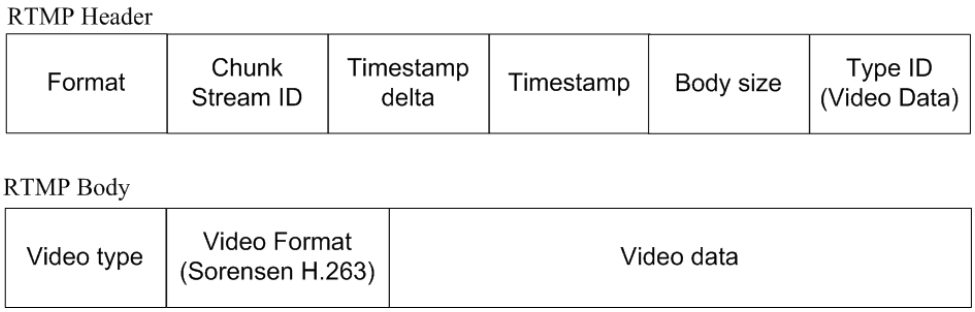


Figure 4.4: The structure of an RTMP Video Data packet.

The TCP packets circled in boxes in Figure 4.3 belong to one special kind of RTMP packet which contains only video data. They share the header of the closest RTMP Video Data packet before it. The size of these packets is either 1230 or 1514 bytes. Therefore all packets from the publisher to FMS contain video data after the connection has been established.

The packet size distributions of four different video resolutions are illustrated in Figure 4.5. The packet size is the payload length of the TCP packet. These plots show that the packets with fixed size of 1176 and 1460 bytes oc-

cupy a significant proportion. This proportion increases with increasing video resolution.

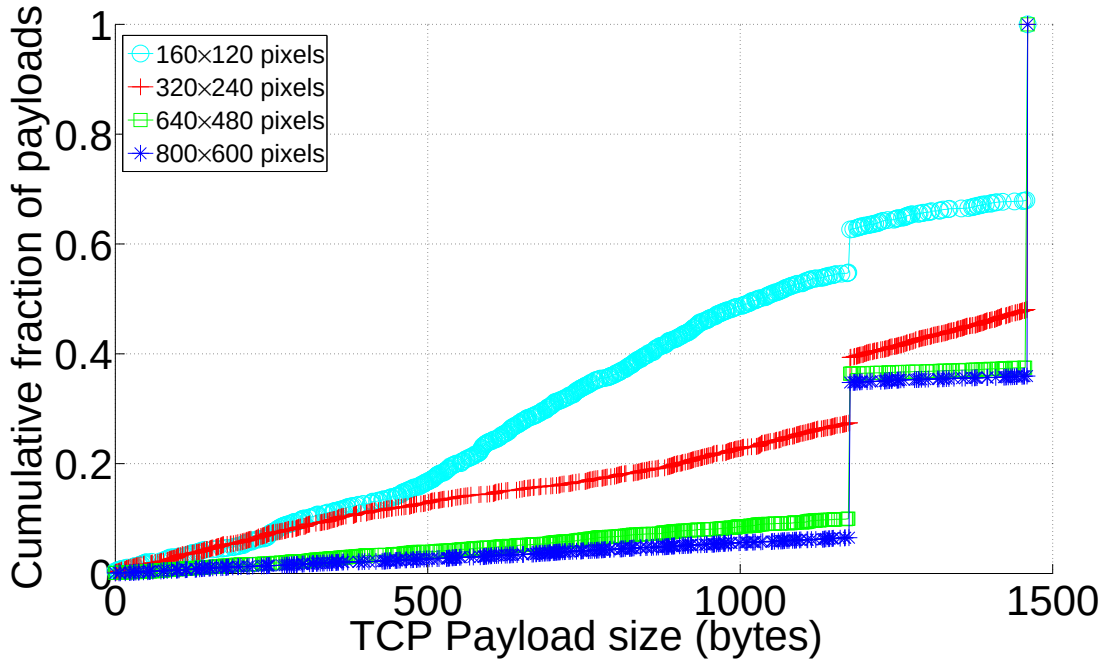


Figure 4.5: The cumulative fraction of packets from the publisher to FMS for different video resolutions.

The proportion of packets containing only video data is quite high. In the RTMP video data packets the number of non-video bytes is much smaller than that of video bytes. Therefore we can assume that all payload in one TCP packet are video data. The estimated transmission bandwidth is listed in Table 4.1.

4.1.2 Receiver

After the RTMFP connection with FMS has been established, the receiver starts to deliver data. The RTMFP connection is a session established between the receiver and FMS. The procedure to establish the connection can be described as:

1. A UDP port on the receiver sends an RTMFP request to port 1935 on FMS and a response is sent back. The response contains the information about redirect port on FMS. The redirect port is 19351 in this case.

2. The UDP port on the receiver communicates with FMS over the redirect port 19351. FMS allocates a UDP port for new session and sends the port number back to the receiver.

Table 4.1: Video transmission bandwidth on the publisher for case one

Video Resolution (pixels)	**Bandwidth (kbps)**
160×120	183
320×240	345
640×480	930
800×600	1017

3. Once the receiver acquires this UDP port number through the redirect port 19351, it starts to establish the session between two UDP ports. The two UDP ports are the peer pair of a session.

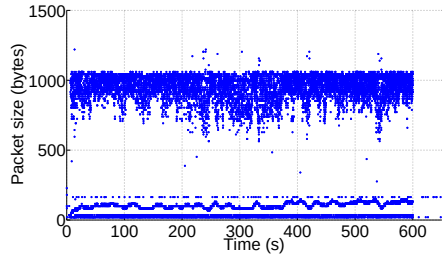
After the session establishment succeeds, the UDP port on the receiver still listens on the redirect port 19351 periodically. This is used to send keep alive packets. The period is about 15 seconds. This value is the same as the *<KeepAliveTime>* setting for the server connection on FMS's configuration file.

The packet size distributions over time for different video resolutions are illustrated in Figure 4.6, Figure 4.7, Figure 4.8 and Figure 4.9 respectively. Only the UDP packets exchanged between two peers are picked out. The packet size is the payload length of UDP packet. These packets captured on the network interface are encrypted using Adobe's Cryptography Profile. They are decrypted in Flash Player. Their structures are still not open even RTMFP has been partly released in [31].

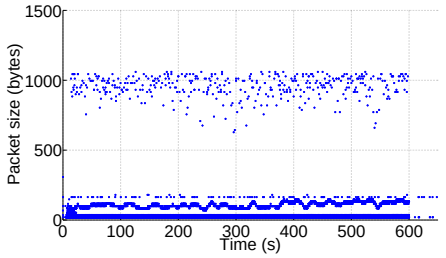
(a) Received packets vs. Time.

(b) Sent packets vs. Time.

Figure 4.6: Packet size distribution over time on one receiver for video resolution 160×120 pixels.

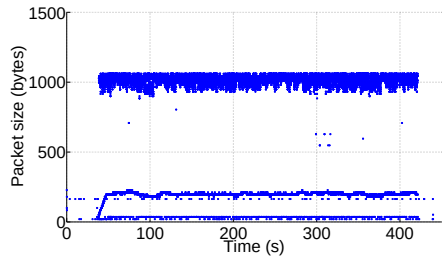


(a) Received packets vs. Time.

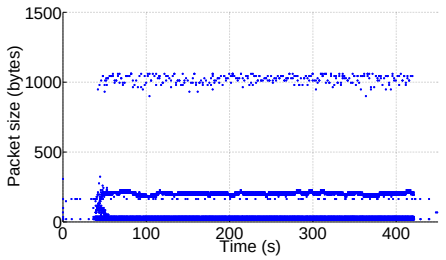


(b) Sent packets vs. Time.

Figure 4.7: Packet size distribution over time on one receiver for video resolution 320×240 pixels.

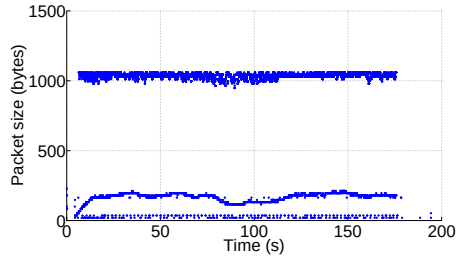


(a) Received packets vs. Time.

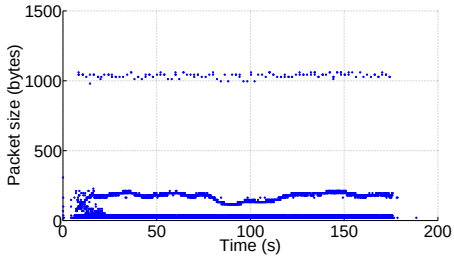


(b) Sent packets vs. Time.

Figure 4.8: Packet size distribution over time on one receiver for video resolution 640×480 pixels.



(a) Received packets vs. Time.



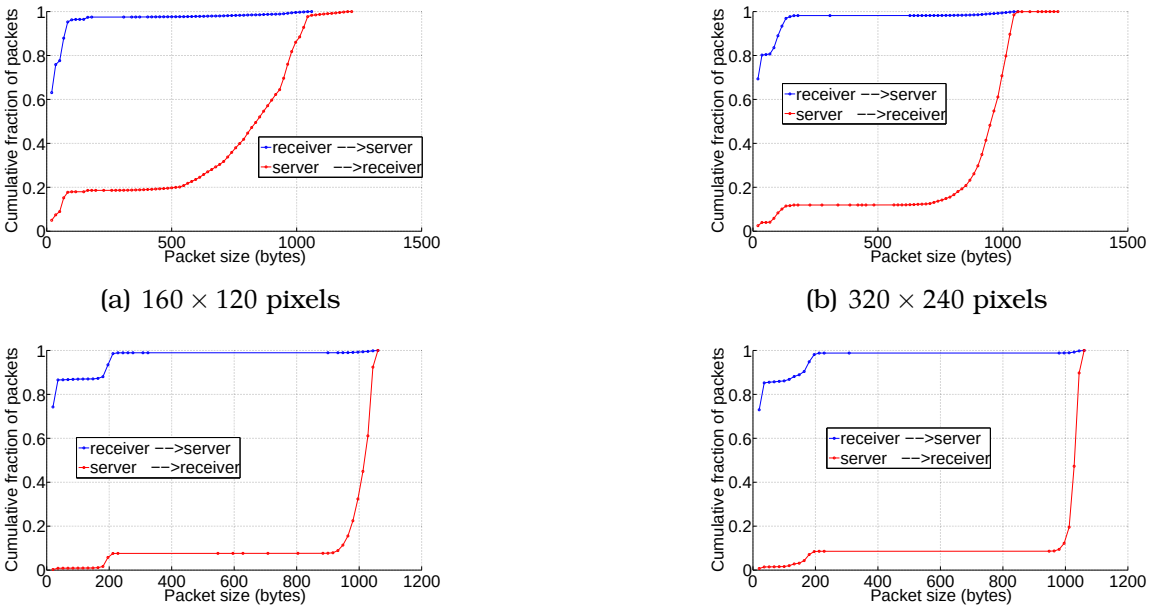
(b) Sent packets vs. Time.

Figure 4.9: Packet size distribution over time on one receiver for video resolution 800×600 pixels.

It is observed that the packet size lies between 20 and 1060 bytes in these measurements. These packets can be categorized into two groups according to their size. The size of small size packets lies between 20 and 250 bytes. The large size packets become more densely distributed around the size of 1000 bytes with increasing video resolution. We guess the large size packets contain video data and the small size packets contain control data. With this assumption, there should be video data sent from the receiver to FMS. This

phenomenon matches with the data delivery method used in the RTMFP P2P group technology.

The packet size distributions for different video resolutions are illustrated in Figure 4.10. The fractions of small size packets among packets indicate the ratios of control overhead. The ratios of control overhead in traffic from FMS to the receiver are listed in Table 4.2. The control overhead accounts for about 10% of traffic except for video resolution 160×120 pixels.



(a) 160×120 pixels

(b) 320×240 pixels

(c) 480×360 pixels

(d) 800×600 pixels

Figure 4.10: Packet size distribution on one receiver for case one.

Table 4.2: Ratio of control overhead on the receiver for case one

Video Resolution (pixels)	**Ratio of control overhead**
160×120	19%
320×240	12%
640×480	8%
800×600	9%

The video transmission bandwidth can be estimated using the large size packets. The results are listed in Table 4.3. Compared with the value in Table 4.1, the video transmission bandwidth on the receiver is about the same as that on the publisher.

Table 4.3: Video transmission bandwidth on the receiver for case one

Video Resolution (pixels)	**Bandwidth (kbps)**
160×120	220
320×240	324
640×480	1104
800×600	948

4.2 Case Two: Two Receivers Within Two Networks

In this measurement scenario, two receivers are running on two computers respectively located in home network and university's wireless network. Those two networks use NAT. The publisher is running on a computer located in university's fixed network. All three networks are not multicast-enabled. The measurement scenario is shown in Figure 4.11. The resolution of published video is 640×480 pixels. The RTMFP group uses fusion multicast. The packets on two receivers are captured using Wireshark.

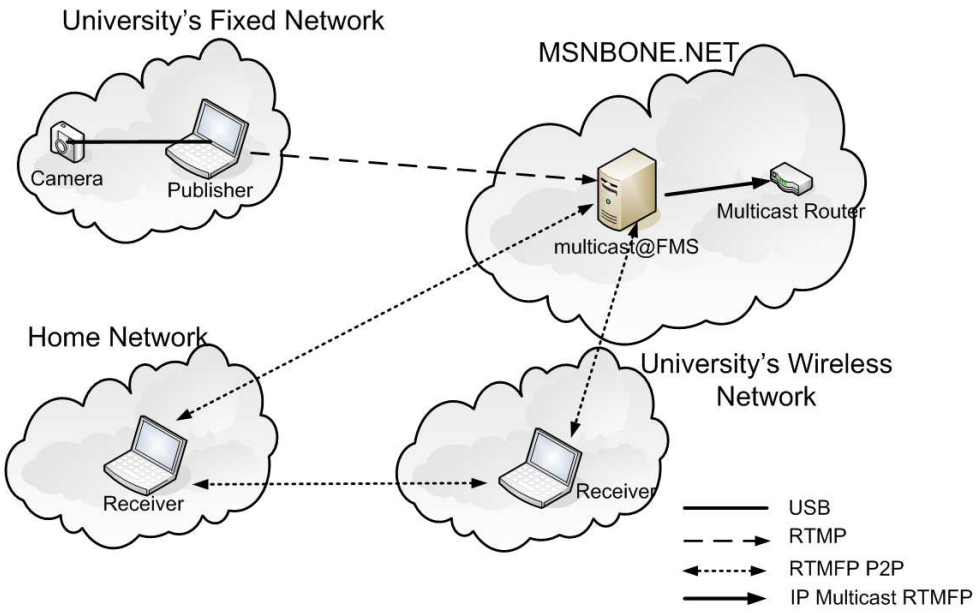


Figure 4.11: Measurement scenario for case two: with two receivers are in two networks. Two receivers and one publisher are running on three computers located on three separate networks.

Each receiver first establishes an RTMFP connection with FMS. Through the redirect port 19351 on FMS, one receiver obtains the information of another receiver. Then the two receivers start to establish session with each other. The connected peers are shown in Figure 4.12. Each peer is one UDP port on the computer. These peers are the members of an RTMFP group.

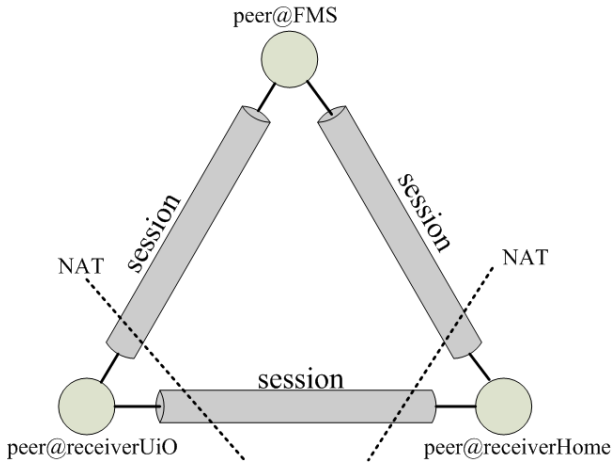
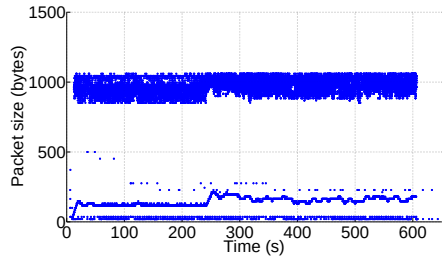


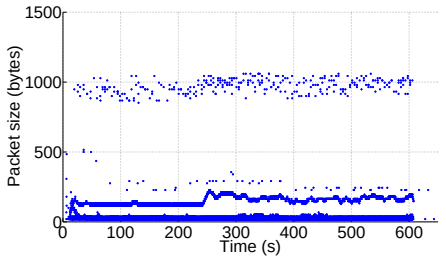
Figure 4.12: Three connected peers.

Two receivers are equal peers in this case. Only the packets captured on the receiver located in home network are analyzed and presented. The packet size distributions over time are shown in Figure 4.13. The packets received from FMS and another peer receiver are shown in Figure 4.13(a) and Figure 4.13(c)

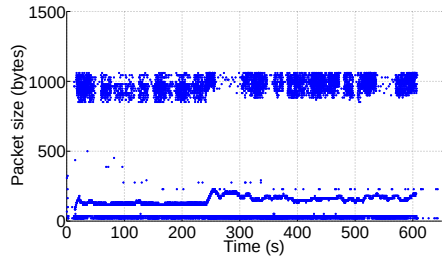
respectively. As shown in Figure 4.13(b) and Figure 4.13(d), the receiver also sends packets to FMS and another peer receiver. There exists exchange of large size packets between the two receivers.



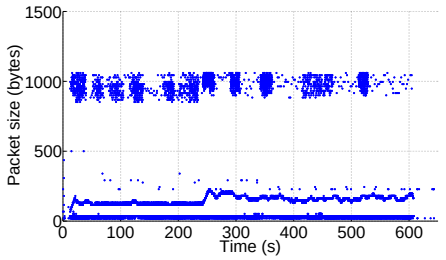
(a) Packets received from FMS.



(b) Packets sent to FMS.



(c) Packets received from another receiver.



(d) Packets sent to another receiver.

Figure 4.13: Packet size distribution over time on one receiver for case two.

The packet size distribution is illustrated in Figure 4.14. It shows that the proportion of large size packets from FMS is larger. This means that the receiver acquires more video data from FMS than that from another peer receiver. FMS provides video source in the RTMFP group.

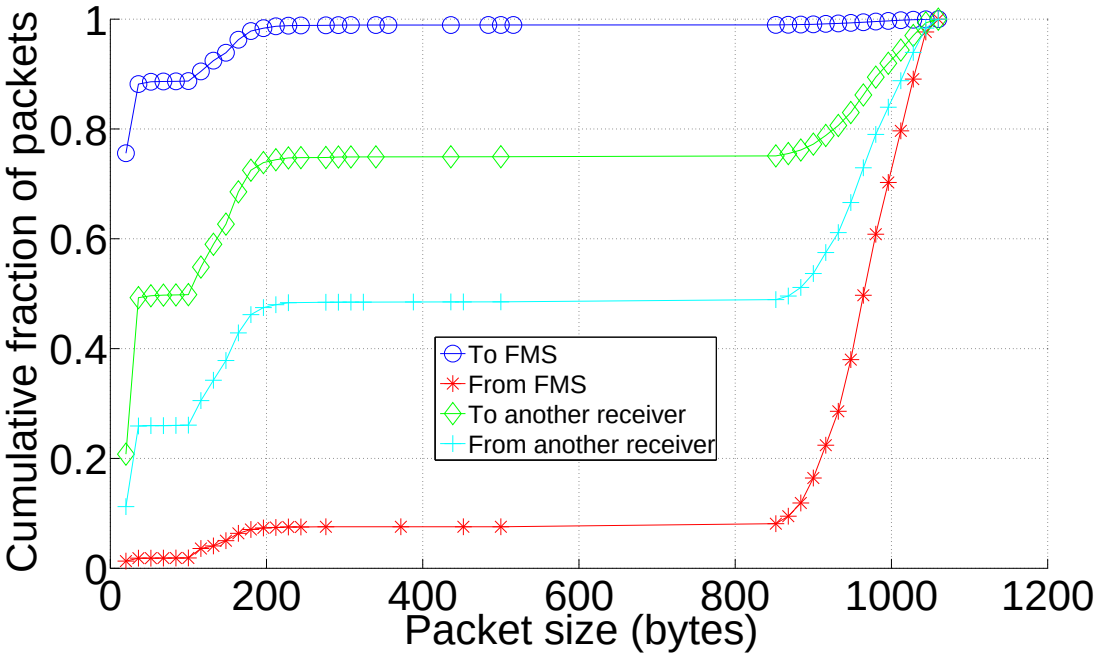


Figure 4.14: Packet size distribution on one receiver for case two.

The video transmission bandwidth on the receivers is estimated using the large size packets. The results are listed in Table 4.4.

Table 4.4: Video transmission bandwidth on the receivers for case two

Source	**Destination**	**Bandwidth (kbps)**
Receiver 1	FMS	**4**
	Receiver 2	**38**
Receiver 2	FMS	**4**
	Receiver 1	**86**
FMS	Receiver 1	**629**
	Receiver 2	**680**

According to the data delivery method used in the RTMFP P2P group, there could exist duplicated video segments on one receiver. However the proportion of the duplicated video segments should be very low. The total video data can

be approximated as the sum of video data from its two sources. Compared to the bandwidth on FMS using unicast, the saved bandwidth on FMS using P2P multicast is calculated in percentage as:

$$percent_{saved\_bandwidth} = \frac{38 + 86}{629 + 86 + 680 + 38} = 8.7\%. \quad (4.1)$$

If there are more receivers in the RTMFP group, one receiver can also acquire video data from more peer receivers. More bandwidth on FMS will be saved with larger number of receivers.

4.3 Case Three: Two Receivers Within Multicast-Enabled Network

In this measurement scenario, two receivers and one publisher are running on three computers located the MNSBONE.NET. This network is multicast-enabled. The measurement scenario is illustrated in Figure 4.15. The resolution of published video is 160×120 pixels. The RTMFP group uses fusion multicast. The packets on one receiver are captured using Wireshark.

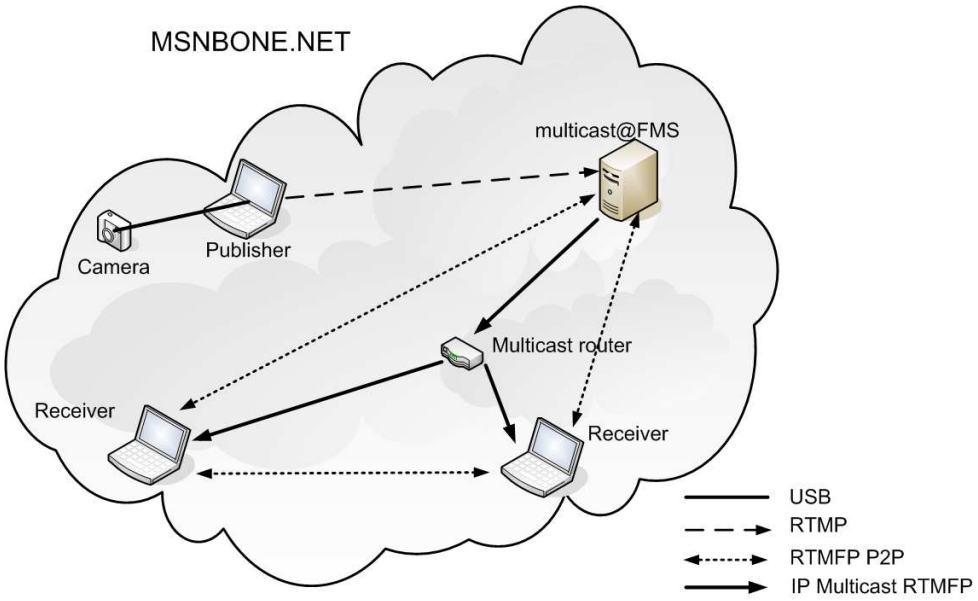
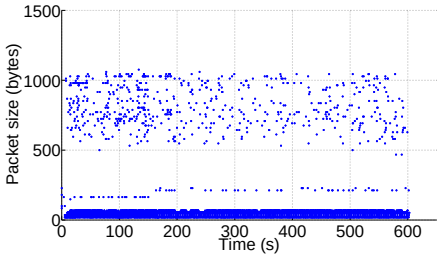
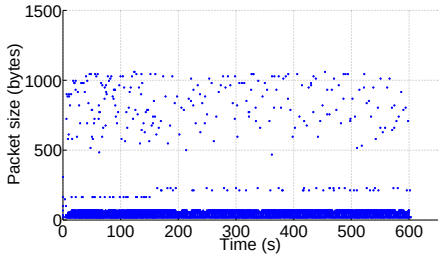


Figure 4.15: Measurement scenario for case three: with two receivers in multicast-enabled network. Two receivers and one publisher are running on three computers located on MNSBONE.NET. This network is multicast-enabled.

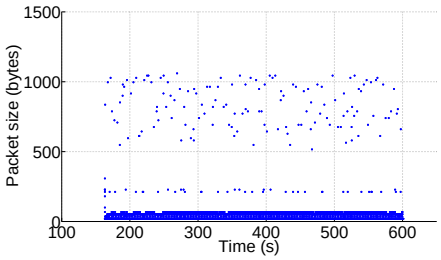
The packet size distributions over time are illustrated in Figure 4.16. One receiver can obtain packets from FMS, another peer receiver and the IP multicast group. The received packets from different sources are illustrated in Figure 4.16(a), Figure 4.16(c) and Figure 4.16(e) respectively. As illustrated in Figure 4.16(b) and Figure 4.16(d), the receiver also sends packets to FMS



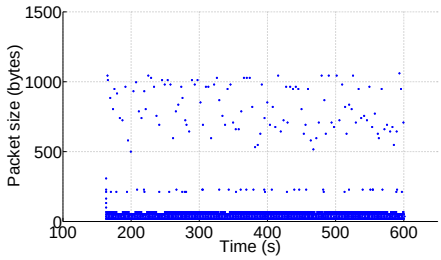
(a) Received packets from FMS.



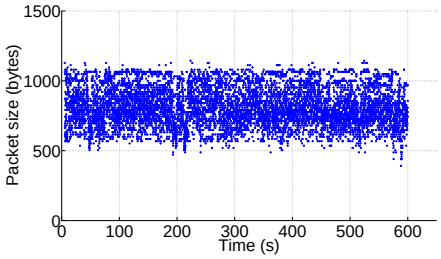
(b) Sent packets to FMS.



(c) Received packets from another receiver.



(d) Sent packets to another receiver.



(e) Received multicast packets.

Figure 4.16: Packet size distribution over time on one receiver for case three.

and another receiver. From Figure 4.16(c) and Figure 4.16(d), we see that the packets exchange with another peer receiver starts a few minutes later. This implies that another peer receiver joins later. As illustrated in Figure 4.16(e), the received IP multicast packets are all of large size.

Similar to the case in Section 4.2, an RTMFP group consists of FMS and two receivers. All the group members are exchanging large size packets with each other. The difference is that the computers with receiver are located in a multicast-enabled network. The large size packets are mainly from IP multicast. P2P multicast is used in the same time as IP multicast, but IP multicast is dominant in the multicast-enabled network.

The packet size distribution is illustrated in Figure 4.17. The packets exchanged among the group members are mainly small size packets for controlling. There exists no small size IP multicast packet. This fact implies that IP multicast has no control data.

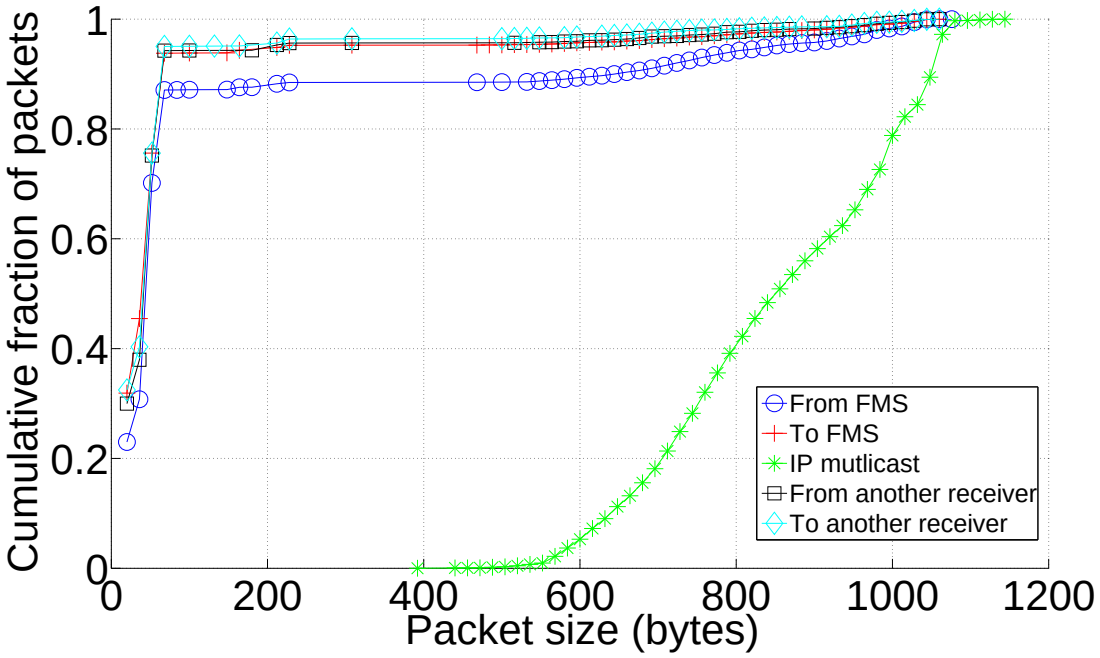


Figure 4.17: Packet size distribution on one receiver for case three.

The transmission bandwidth to receive video is calculated using the large size packets. The results are listed in Table 4.5. We see that video data is mainly delivered using IP multicast. The bandwidth using P2P multicast only counts for about 5% of that using IP multicast. If a new receiver in the multicast-enabled network joins the RTMFP group, it will obtain video data mainly using IP multicast. The bandwidth on FMS will not double, but only increases around 5%. The bandwidth on FMS does not increase proportionally with increasing the number of receivers in multicast-enabled networks.

Table 4.5: Transmission bandwidth to receive video for case three

Type of multicast	**Bandwidth (kbps)**
P2P multicast from FMS	**7**
P2P multicast from another receiver	**2**
IP multicast	**140**

4.4 Some Practical Problems in Experiments

In order to involve as many receivers as possible for measurements, we have tried to perform the experiments in other scenarios. Some experiments have similar results, such as a measurement scenario in which two receivers are in home network. The results are very similar to those shown in Section 4.2.

Some practical problems have limited our experiments.

Some networks block the RTMFP connection or forbid the use of uplink bandwidth. The receiver can not run on computers located in such networks. There are many computers available in university, but the university's fixed network blocks RTMFP connection. Therefore we have very limited number of available computers.

The publisher and receiver applications can only be running in web browsers supporting Flash Player 10.1 and later. It is slow to open web browsers remotely. Some server OS does not support the newer version of Flash Player. Therefore it is difficult to perform global experiments which involve the computers in different geographic locations.

Chapter 5

Conclusion

In this chapter, the work is first summarized in Section 5.1. Then in Section 5.2, the contributions are presented. Finally some future work is proposed in Section 5.3.

5.1 Summary

In this thesis, a global multicast demonstrator for live video streaming has been implemented on Adobe's Flash platform.

In Chapter 2, we have first investigated the multicast communication mechanisms. By comparing IP multicast and P2P multicast, we have figured out that the combination of both multicast technologies provides an efficient global communication mechanism. We have also described several media streaming systems and the related streaming technologies, with focusing on Adobe's media streaming systems.

In Chapter 3, we have presented the technologies and tools on Adobe's Flash platform. We have installed and configured Adobe's Flash Media Server on the server machine located on the multicast-enabled network. We have developed the client-side Flash applications for publishing, receiving and playing video. The FMS and client-side Flash applications compose our global multicast demonstrator for live video streaming.

In Chapter 4, we have performed a few experiments using the prototype demonstrator in real world Internet. The measurement results have been presented and discussed. We have also listed some practical problems we have met in experiments.

5.2 Contributions

In this work, we have implemented a prototype demonstrator on Adobe's Flash platform for live video streaming. This demonstrator utilizes both IP multicast and P2P multicast. It can demonstrate live video streaming in real network

environments. We have performed a series of experiments using this prototype demonstrator. It has been concluded that the use of multicast in video streaming can reduce the load on the server. We have also concluded that P2P multicast provides a possible solution for global video streaming with increased scalability.

5.3 Future Work

More experiments could be performed for further analysis. Firstly, the experiments involving more receivers are interested to further investigate the scalability of this demonstrator. Our experiments have only involved limited number of computers due to some practical problems.

Secondly, experiments involving more receivers on computers located in different places are interested. Our experiments have been performed mostly on computers locally due to difficulties in running client applications on remote computers.

Finally, some improvements can be performed in the prototype demonstrator to make it really global. In addition to RTMFP, Adobe's Flash platform also supports other streaming protocols such as RTMP. RTMP is TCP-based and allowed in most network environments. The receiver in the prototype demonstrator can shift to RTMP connection if the attempt to establish the RTMFP connection fails. This modification will make the demonstrator more global for live video streaming. based on RTMP uses unicast. Such modification has utilized unicast and deviated from our goal using multicast.

Bibliography

[1] Server-Side ActionScript Language Reference for the Flash Media Server 4.5. Adobe Systems Incorporated, Dec. 2012.

[2] ActionScript 3.0 Reference for the Adobe Flash Platform. Adobe Systems Incorporated, Apr. 2013.

[3] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), Sept. 2009.
http://www.ietf.org/rfc/rfc5681.txt.

[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. *SIGCOMM Comput. Commun. Rev.*, 32(4):205–217, Aug. 2002.
http://doi.acm.org/10.1145/964725.633045.

[5] M. Castro, P. Druschei, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. CoolStreaming/DONet: a Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 298–313, 2003.
http://doi.acm.org/10.1145/945445.945474.

[6] Y.-H. Chu, A. Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early experience with an internet broadcast system based on overlay multicast. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 12–12, 2004.
http://dl.acm.org/citation.cfm?id=1247415.1247427.

[7] Y.-H. Chu, S. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *Selected Areas in Communications, IEEE Journal on*, 20(8):1456–1471, 2002.

[8] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2011-2016. WHITE PAPER, May 2012.

[9] S. E. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Trans. Comput. Syst.*, 8(2):85–110, May 1990.
http://doi.acm.org/10.1145/78952.78953.

[10] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *Network, IEEE*, 14(1):78–88, 2000.

[11] S. Floyd. Congestion Control Principles. RFC 2914, Sept. 2000. http://www.ietf.org/rfc/rfc2914.txt.

[12] A. Ganesh, A. M. Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols. *Computers, IEEE Transactions on*, 52(2):139–149, 2003.

[13] G. Grossman and E. Huang. ActionScript 3.0 Overview. Adobe Developer Connection, June 2006. http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html.

[14] C. Hedrick. Routing Information Protocol. RFC 1058 (Historic), June 1988. http://www.ietf.org/rfc/rfc1058.txt.

[15] X. Hei, Y. Liu, and K. Ross. IPTV over P2P streaming networks: the mesh-pull approach. *Communications Magazine, IEEE*, 46(2):86–92, 2008.

[16] M. Kwon and S. Fahmy. Topology-aware overlay networks for group communication. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 127–136, 2002. http://doi.acm.org/10.1145/507670.507688.

[17] B. Li, S. Xie, G. Keung, J. Liu, I. Stoica, H. Zhang, and X. Zhang. An empirical study of the coolstreaming+ system. *Selected Areas in Communications, IEEE Journal on*, 25(9):1627–1639, 2007.

[18] B. Li, S. Xie, Y. Qu, G. Keung, C. Lin, J. Liu, and X. Zhang. Inside the New Coolstreaming: Principles, Measurements and Performance Implications. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1031–1039, 2008.

[19] Y. Liu, Y. Guo, and C. Liang. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Networking and Applications*, 1(1):18–28, 2008. http://dx.doi.org/10.1007/s12083-007-0006-y.

[20] N. Magharei and R. Rejaie. PRIME: Peer-to-Peer Receiver-drIven MEsh-Based Streaming. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1415–1423, 2007.

[21] N. Magharei, R. Rejaie, and Y. Guo. Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In *INFOCOM 2007. 26th*

IEEE International Conference on Computer Communications. IEEE, pages 1424–1432, 2007.

[22] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: eliminating trees from overlay multicast. In *Proceedings of the 4th international conference on Peer-to-Peer Systems*, pages 127–140, 2005. http://dx.doi.org/10.1007/11558989_12.

[23] H. Parmar and M. Thornburgh. RTMP Specification 1.0. Adobe Systems Incorporated, Dec. 2012.

[24] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), Sept. 2000. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665, 6878
http://www.ietf.org/rfc/rfc3261.txt.

[25] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (INTERNET STANDARD), July 2003. Updated by RFCs 5506, 5761, 6051, 6222
http://www.ietf.org/rfc/rfc3550.txt.

[26] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *MultiMedia, IEEE*, 18(4):62–67, 2011.

[27] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. *SIGCOMM Comput. Commun. Rev.*, 34(4):107–120, Aug. 2004.
http://dx.doi.org/10.1016/j.comnet.2005.07.016.

[28] T. Stockhammer. Dynamic adaptive streaming over HTTP –: standards and design principles. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 133–144, 2011.
http://doi.acm.org/10.1145/1943552.1943572.

[29] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, Aug. 2001.
http://doi.acm.org/10.1145/964723.383071.

[30] M. Thornburge. Advanced P2P with RTMFP: Tips and Tricks. Presentation from MAX 2011 Develop, Oct. 2011.

[31] M. Thornburge. Adobe's Secure Real-Time Media Flow Protocol. IETF Internet-Draft, Feb. 2013. Expires: August 18, 2013
http://tools.ietf.org/html/draft-thornburgh-adobe-rtmfp-04.

[32] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. RFC 1075 (Experimental), Nov. 1988. http://www.ietf.org/rfc/rfc1075.txt.

[33] B. Zhang, W. Wang, S. Jamin, D. Massey, and L. Zhang. Universal IP multicast delivery. *Comput. Netw.*, 50(6):781–806, Apr. 2006. http://dx.doi.org/10.1016/j.comnet.2005.07.016.

[34] X. Zhang, J. Liu, B. Li, and T. Yum. CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2102–2111, 2005.